UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

SOFTWARE DE PROGRAMACIÓN PARA ROBOT BRACCIO

Memoria y Anexos

Autor: Tania Angelica Castro Tapias
Director: Sebastian Tornil Sin

Convocatoria: 2018

Resum

El present projecte de final de grau té com a objectiu principal crear un software de programació o aplicació a través de la qual es pugui controlar y programar un braç robòtic comercial, en aquest cas el Tinkerkit Braccio Robot d'Arduino.

Aquesta aplicació permetrà fer servir el braç robòtic de manera similar al control per consola de programació dels robots manipuladors industrials, és a dir, es podrà fer moure cada articulació a voluntat i es podran memoritzar posicions determinades del robot, amb les quals posteriorment es podran realitzar, executar i guardar petits programes per tal de fer moure el braç de determinada manera.

L'aplicació consta de dos parts: la base és la programació del microcontrolador de la placa de desenvolupament Arduino Uno connectat al braç, el qual ha de processar les ordres que rebi des d'una altra plataforma o interfície amb el usuari, que conforma la segona part important i que pot tractar-se d'una aplicació mòbil, com és el cas d'aquest projecte.

Resumen

El presente trabajo de final de grado tiene como objetivo principal crear un software de programación o aplicación a través de la cual se pueda controlar y programar un brazo robótico comercial, es este caso el Tinkerkit Braccio Robot de Arduino.

Esta aplicación permitirá usar el brazo robótico de manera similar al control por consola de programación de los robots manipuladores Industriales, es decir, se podrá mover cada articulación a voluntad y se podrán memorizar posiciones determinadas del robot, con las cuales posteriormente se podrán realizar, ejecutar y guardar pequeños programas con el fin de mover el brazo de determinada manera.

La aplicación consta de dos partes: la base es la programación del microcontrolador de la placa de desarrollo Arduino Uno conectado al brazo, que ha de procesar las órdenes que reciba desde otra plataforma o interfaz con el usuario, que conforma la segunda parte importante y que se puede tratar de una aplicación móvil, como es el caso de este trabajo.

Abstract

The main goal of this final degree project is creating a programming software or application with which programming and controlling a commercial robotic arm, in this case Arduino's Tinkerkit Braccio Robot, is enabled.

This application will enable using the robotic arm in a similar way to the programming console control of the industrial manipulator robots. That means the robotic arm will be able to move as wished and determinate robot positions will be able to be saved and used later in small programs, which the user will be able to create, execute and save in order to move the robotic arm in a determinate way.

The application has two parts: the base is the programming of the Arduino Uno microcontroller board that has to process received orders from another platforms or user interfaces, which is the second important part and can be, e.g., a mobile app, as in this project.

Índice

Índice de ilustraciones

Índice de tablas

1. Introducción

Este trabajo de final de grado (TFG) está orientado a la programación del microcontrolador de la placa de desarrollo Arduino Uno conectado al brazo robótico Tinkerkit Braccio Robot (robot Braccio en adelante) con el fin de hacer posible su movimiento y programación desde otra aplicación que hace la función de interfaz de usuario y que, en este caso, se trata de una aplicación móvil, cuya programación también forma parte del presente trabajo.

El hecho de poder controlar un robot desde un móvil y poderlo hacer de forma similar a como se haría con los robots manipuladores industriales, hace que la introducción al mundo de la automatización sea más cercana, barata, amigable y cree interés.

1.2. Origen del trabajo

Este trabajo se origina en la intención de mejorar el uso que se le puede dar al robot Braccio, ya que la librería que se proporciona en la web de Arduino contiene dos instrucciones básicas, que se explicarán más adelante y las cuales, entre otros inconvenientes, hacen que la programación y el uso del robot Braccio sea algo tediosa y muy básica.

Por otra parte, se podría llegar a conseguir un método de aprendizaje de programación de robots más barata y accesible a más personas.

1.3. Motivación

La inclinación del autor hacia la automatización y robótica industrial cultivada a lo largo de los últimos dos años de carrera, ha originado el interés por este TFG, que implica realizar una mejora de un producto comercial de propósito educativo y de ocio como lo es el Tinkerkit Braccio Robot, de manera que éste se pueda usar de forma similar a un robot manipulador industrial, como el Mitsubushi RV-2AJ, presentado en la asignatura “Robótica Industrial y Visión por Computador” durante el quinto cuatrimestre del Grado en Ingeniería Electrónica Industrial y Automática de la Escuela de Ingeniería de Barcelona Este (EEBE, UPC).

Ilustración 1.1. Tinkerkit Braccio Robot [1]

Ilustración 1.2. Mitsubishi RV-2AJ [2]

Otra de las motivaciones para realizar un trabajo que se basa en la programación es también la predisposición y ganas de aprender nuevos lenguajes y entornos de programación.

1.4. Requerimientos previos

Puesto que la programación es el pilar central de este trabajo, los conocimientos previos sobre algún lenguaje son importantes, de la misma manera que la predisposición a aprender nuevos y, por lo tanto, la capacidad de aprender de forma autónoma jugará un papel importante. En el mejor de los casos, un buen conocimiento de Java y C++ puede hacer que esta tarea se haga muy fluida.

Por otra parte, los conocimientos sobre electrónica son también relevantes, ya que con éstos se pueden arreglar posibles fallos de hardware o modificar el mismo, en caso de que sea necesario.

Además de los conocimientos técnicos previos, se trabajarán y reforzarán las habilidades de planificación del trabajo, la capacidad de dividir los problemas en pequeñas partes para así facilitar su resolución y el uso solventes de los recursos de la información.

En resumen, se podría decir que todos los conocimientos adquiridos durante el Grado en Ingeniería Electrónica Industrial y Automática han sido relevantes en la realización del presente trabajo.

1.5. Objetivos del trabajo

El principal objetivo y el más básico es crear una estructura software en el microcontrolador que permita la interacción entre una aplicación externa cualquiera y el robot Braccio, es decir, una estructura que interprete y ejecute las órdenes referentes al movimiento del brazo recibidas a través de algún medio, como por ejemplo, un módulo de Bluetooth conectado al microcontrolador, como es el caso de este trabajo. Para comprobar que ésta funciona, la realización de una aplicación externa también será necesaria, la cual se tratará de una aplicación móvil cuyo desarrollo también se incluirá en este trabajo.

Aquello que se ha de poder interpretar en dicha estructura son órdenes codificadas de: movimiento de articulaciones del robot, guardado de posiciones de articulaciones y programación de movimientos simples, es decir, movimientos consecutivos previamente definidos.

1.6. Alcance del trabajo

Este trabajo es de alcance variable. Esto significa que a medida que se alcancen objetivos y según el tiempo disponible, se pueden realizar más tareas. En el subapartado anterior se presentó el objetivo principal, es decir, aquel objetivo mínimo que se ha de cumplir y que consta de:

- Mover el robot desde la propia interfaz del entorno de desarrollo integrado (IDE) de Arduino.
- Establecer la comunicación con el móvil.
- Mover el robot desde el móvil.
- Guardar posiciones en el móvil y en el microcontrolador.
- Crear la posibilidad de programar los movimientos del robot desde el móvil.

Una vez alcanzados estos objetivos, se presentan las siguientes propuestas de mejora y ampliación:

- Instrucciones de movimiento de punto a punto con trayectoria lineal.
- Instrucciones de movimiento con trayectorias circulares.
- Instrucciones de variación de velocidad.
- Diseño y montaje de una interfaz con la que poder usar las diferentes funciones implementadas en el microcontrolador, sin necesidad de estar este último conectado al móvil o el ordenador.
- Implementación de una aplicación para PC.

- Programación de un juego que haga uso de todo lo anterior.
- Generalizar el código para cualquier número de articulaciones, ya que el robot Braccio admite modificaciones de su estructura.
- Implementar el uso de sensores.
- Implementar instrucciones de control de flujo.
- Añadir funciones con visión artificial.

2. Antecedentes

Antes de abordar el desarrollo de este trabajo, hacen falta un conjunto de conceptos generales sobre los robots, los cuales se describen en esta sección.

2.1. Introducción a la robótica

La primera vez que se usó el término "robot" fue en la obra de teatro checa "Rossum's Universal Robots" de Karel Čapek en 1921 en el Teatro Nacional de Praga, en la cual se creaban humanos sintéticos para aligerar la carga de trabajo de los humanos. El término fue ideado por el hermano del autor, el cual se basó en la palabra checa robota, que significa trabajo, en general, de la servidumbre.

De forma parecida a la situación que plantea la obra teatral, en un principio, los robots fueron creados para sustituir a los humanos en trabajos específicos, ya que carecen de inteligencia, que pueden superar sus capacidades físicas o que pueden resultar peligrosos. Este objetivo añadido al interés por abaratar costes y reducir el tiempo de producción en las industrias, resultó en una gran revolución de la robótica en el mundo industrial, hasta tal punto que, actualmente, son indispensables en la automatización industrial.

Junto con el paralelo desarrollo de otros campos como la electrónica y la informática, las características de los robots fueron mejorando, de manera que su uso empezó a salir del ámbito industrial, como se describirá a continuación en la clasificación de los robots.

2.2. Tipos de robots

Según la norma ISO (Organización Internacional para la Estandarización) 8373, revisada por última vez en 2012, los robots se clasifican en dos grandes grupos:

- Robots industriales: Según la norma ISO 8373, un robot industrial es un manipulador multipropósito, reprogramable y controlado automáticamente, de tres o más ejes, que puede estar fijo o móvil para su uso en aplicaciones de automatización industrial y que, según su estructura mecánica, se clasifican en:
    - Robots lineales:
        - Robots cartesianos: poseen tres articulaciones prismáticas que, al ser perpendiculares, coinciden con el sistema de coordenadas cartesiano.
        - Robots "Gantry": consisten en robots manipuladores montados en un sistema suspendido que permite su movimiento a través de un plano horizontal.

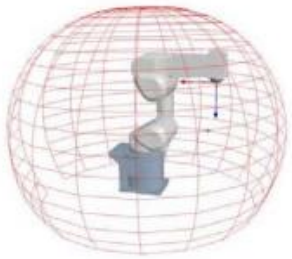
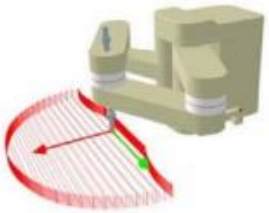
Ilustración 2.1. Robot "Gantry"[8]

- Robots SCARA: se trata de robots con dos ejes rotacionales y uno lineal, paralelos entre ellos. Puede tener un eje adicional para orientar el extremo.
- Robots Articulados: consta de tres ejes rotacionales, uno perpendicular a la base y los otros dos paralelos entre sí y perpendiculares al primer eje.
- Robots paralelos o Delta: son robots formados por articulaciones prismáticas o rotatorias concurrentes (cadena cinemática cerrada).
- Robots cilíndricos: están formados por dos ejes prismáticos perpendiculares y un eje de rotación vertical sobre la base formando así un sistema de coordenadas cilíndrico.
- Otros:
    - Robots "Dual-arm": estructuras formadas por dos brazos articulados.

Ilustración 2.2. Robot "Dual-Arm"[5]

En la siguiente tabla se ilustran los tipos de robots anteriormente descritos.

Tabla 2.1. Tipos de robot industriales, clasificados por su estructura mecánica, y de los cuales se detalla el área de trabajo y se muestra un ejemplo real [5].

Principle	Kinematic Structure	Photo
Articulated Robot		
SCARA Robot		
SCARA Robot		
Cartesian Robot		
Parallel Robot		

- Robots de servicios: Son aquellos robots, cuyas aplicaciones se encuentran fuera del entorno industrial y que se destinan a proporcionar algún tipo de servicio a los humanos. Según su definición en la norma ISO, requieren de cierto grado de autonomía, yendo desde una autonomía parcial hasta una total autonomía, es decir, con o casi sin interacción humano-robot, ya que, aunque se les denomine como robots con total autonomía, se ha de aclarar que han de tener un cierto grado de interacción con un operador. Dentro de este grupo, se pueden diferenciar dos tipos:
    - Robots de servicios para el uso personal: El usuario de este tipo de robots no requiere conocimientos técnicos previos sobre el robot para poderlo usar. La Federación Internacional de Robótica (IFR) distingue los siguientes tipos:
        - Robots de tareas domésticas: Con el objetivo de facilitar y agilizar las tareas domésticas, que dentro de una sociedad frenética como la actual representan todo un inconveniente, estos robots pueden ser, por ejemplo, limpiasuelos, limpiacristales, cortacésped o limpiafondos.

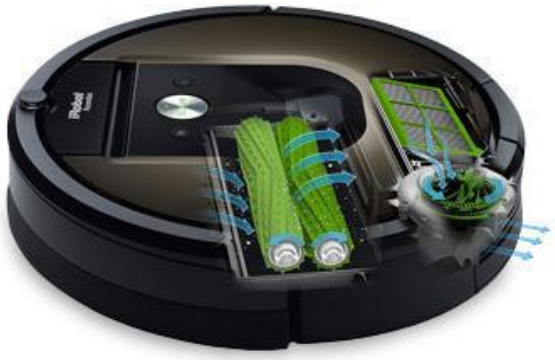


Ilustración 2.3. iRobot Roomba [9]

Ilustración 2.4. Winbot W710 [10]

Ilustración 2.5. Robot limpiafondos Dolphin Supreme M5 [11]

- Robots de entretenimiento: No cumplen una tarea específica. Pueden tratarse de juguetes, robots educativos, kits de montaje de robots o "robot rides".

Ilustración 2.6. Robot Zowi de BQ [12]

Ilustración 2.7. KUKA "*Coaster*" [13]

- Robots de asistencia a los discapacitados: A través de las continuas mejoras en las características de los robots, actualmente, se han desarrollado y se desarrollan aplicaciones que ayuden en el día a día de personas con algún tipo

de minusvalía. Algunos ejemplos de este tipo de robots son las sillas de ruedas robotizadas, robots de asistencia personal capaces de interactuar con el usuario, plataformas elevadoras salvaescaleras o exoesqueletos.

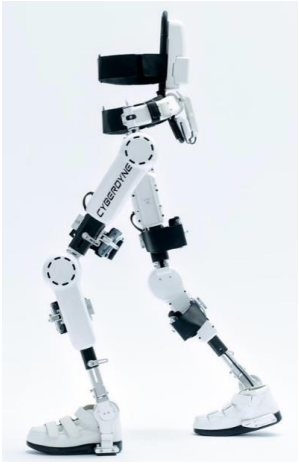


Ilustración 2.8. HAL para uso médico [14]

- Transporte personal – AGV (Automated Guided Vehicles) para personas.
- Seguridad y vigilancia doméstica.

- Robots de servicios para el uso profesional: En este caso, al contrario que el anterior, es necesario que el usuario u operario posea algún tipo de formación previa, de manera que sea capaz de iniciar el robot o sistema robótico, monitorizarlo y pararlo. Este tipo de robots se pueden encontrar en diversos ámbitos como la agricultura, la ganadería, la minería, la limpieza profesional o la medicina.

Ilustración 2.9. DaVinci."*Surgery*" [16]

Esta clasificación se puede ver de forma más detallada en la Tabla 2.2.

Tabla 2.2. Clasificación de los robots de servicios para el uso doméstico y personal [15]

Section I	Types of robots: Service robots for personal/domestic use
1-6	Robots for domestic tasks
1	Robot companions/assistants/humanoids
2	Vacuuming, floor cleaning
3	Lawn-mowing
4	Pool cleaning
5	Window cleaning
6	Others
7-10	Entertainment robots
7	Toy/hobby robots
8	Multimedia/remote presence
9	Education and research
10	Others
11-13	Elderly and handicap assistance
11	Robotized wheelchairs
12	Personal aids and assistive devices
13	Other assistance functions
14	Personal transportation (AGV for persons)
15	Home security & surveillance
16	Other Personal / domestic robots

Tabla 2.3. Clasificación de los robots de servicios destinados al uso profesional [15].

Section II	Types of robots: Service robots for professional use
17-23	Field robotics
17	Agriculture
18	Milking robots
19	Other robots for livestock farming
20	Forestry and silviculture
21	Mining robots
22	Space robots
23	Other field robotics
24-28	Professional cleaning
24	Floor cleaning
25	Window and wall cleaning (incl. wall climbing robots)
26	Tank, tube and pipe cleaning
27	Hull cleaning (aircraft vehicles etc.)
28	Other cleaning tasks
29-31	Inspection and maintenance systems
29	Facilities, plants
30	Tank, tubes, pipes and sewers
31	Other inspection and maintenance systems
32-35	Construction and demolition
32	Nuclear demolition & dismantling
33	Building construction
34	Robots for heavy/civil construction
35	Other construction and demolition systems
36-39	Logistic systems
36	Automated guided (AGV) vehicles manufacturing environments
37	AGVs non-manufacturing environments (indoor)
38	Cargo handling, outdoor logistics
39	Other logistic systems
40-43	Medical robotics
40	Diagnostic systems
41	Robot assisted surgery or therapy
42	Rehabilitation systems
43	Other medical robots
44-46	Rescue & security applications
44	Fire and disaster fighting robots
45	Surveillance / security robots
46	Other rescue and security robots
47-50	Defense applications
47	Demining robots
48	Unmanned aerial vehicles
49	Unmanned ground based vehicles
50	Unmanned underwater vehicles
51	Other defense applications
52	Underwater systems (civil / general use)
53	Powered Human Exoskeletons
54	Unmanned aerial vehicles (general use)
55	Mobile Platforms in general use
56-60	Underwater systems (civil / general use)
56	Hotel & restaurant robots
57	Mobile guidance, information robots
58	Robots in marketing
59	Robot joy rides
60	Others (i.e. library robots)
61	Other professional service robots not specified above

2.3. Programación de robots

Programar, en este contexto, quiere decir describir un conjunto de acciones consecutivas que el robot realizará con el fin de completar una determinada tarea. Estas acciones pasan por el movimiento del elemento terminal a puntos previamente definidos, describiendo o no una trayectoria, con el objetivo de manipular o procesar un objeto, con la ayuda o no de elementos externos que permitan al robot interactuar con su entorno.

En la programación se distinguen tres tipos o métodos, los cuales se presentarán a continuación.

2.3.1. Programación por guiado u "*on-line*"

Como su nombre indica, consiste en que el operario mueve el robot a la vez que éste graba esa serie de movimientos para, posteriormente, reproducirlos. Por esta razón, se podría decir que el usuario guía al robot.

Por tanto, se pueden distinguir dos fases diferenciadas: la primera es en la que se le enseña al robot qué movimientos ha de realizar y de qué manera y, la segunda, en la que el usuario le pide al robot que se mueva reproduciendo de forma autónoma las acciones aprendidas en la fase anterior.

Según la manera en la que se le enseñen los movimientos al robot, la programación por guiado se podría clasificar en:

- "Teach pendant" o guiado activo: El operario mueve el robot desde una consola de programación, como las de la Ilustración 2.10, y graba las posiciones o configuraciones del robot que crea que son de interés.

Ilustración 2.10. Ejemplos de consola de programación para robots industriales [17]

- "Lead Through" o guiado pasivo: El usuario usa su propia fuerza para mover el robot, el cual puede tener los actuadores desconectados, por lo que el usuario ha de cargar con la resistencia que ofrezca la estructura a ser desplazada, o conectados. En este último caso, el usuario puede

guiar al robot a través de un maniquí con la misma configuración cinemática, de manera que al moverlo, el robot lo imite, o directamente, usando un robot con sensores que reconocen que se está intentando mover su extremo de cierta manera y así, adapta sus actuadores para moverse solidariamente, con lo cual, el usuario apenas se ha de esforzar en moverlo.

Ilustración 2.11. Ejemplo de "Lead Through Programming" con los actuadores desconectados *[19]*

Es fácil reconocer que para poder llevar a cabo este tipo de programación, es necesario contar con la presencia de un robot totalmente operativo y en funcionamiento durante el procedimiento, lo que puede suponer un inconveniente, por no poder poner en la línea de producción y suponer cierto riesgo de seguridad al tener que interactuar con él. Por otra parte, el hecho de poder describir trayectorias complejas y obtener puntos específicos sin la necesidad de expresarlo en forma de texto es una gran ventaja a tener en cuenta, ya que ahorra tiempo, se requiere de menos conocimientos para llevarlo a cabo y elimina errores por una incorrecta calibración. En contrapartida, resulta complicado documentar las acciones del robot, modificar el programa e introducir acciones sincronizadas u condicionadas por el entorno (sensores y otras máquinas).

2.3.2. Programación textual u "*off-line*"

Se trata de introducir en forma de texto una secuencia de instrucciones que describan los movimientos que ha de realizar el robot. Estas instrucciones son predefinidas por el fabricante, que facilita un entorno y un lenguaje de programación de muy alto nivel, ya que no es necesario que el usuario posea conocimientos sobre la lógica detrás del movimiento de un robot.

En este contexto, existen tres niveles de programación textual:

- Nivel tarea: En el programa se describe el objetivo a conseguir, por ejemplo, "pintar coche" o "ensamblar engranaje".
- Nivel objeto: Se precisan las operaciones a realizar sobre las partes que se han de manipular o procesar, estableciendo, si es necesario, la relación entre ellas. Por ejemplo: "situar PIEZA_A sobre PIEZA_B, haciendo coincidir LADO_A1 con LADO_B3".

- Nivel robot: Es el nivel más usado y en él se especifican cada una de las acciones que ha de realizar el robot para conseguir el objetivo deseado. Cabe destacar que incluye instrucciones de manejo de entradas (sensores), salidas (señales hacia actuadores) y comunicaciones (con otras máquinas o robots). Por ejemplo: “Pinza abrir”, “Mover linealmente posicion5”.

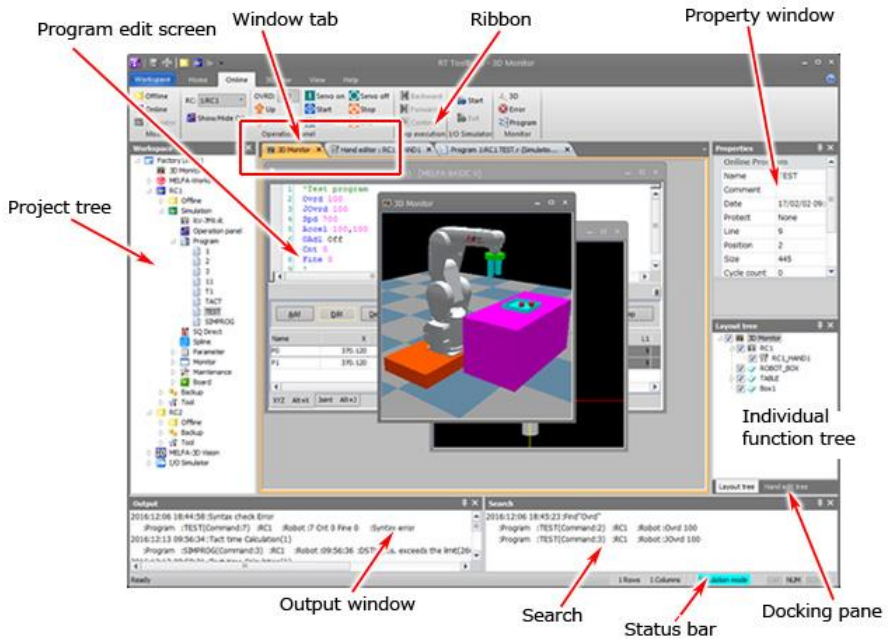


Ilustración 2.12. Ejemplo de entorno de programación para robots [20]

Este método permite crear rutinas de mayor complejidad y precisión sin requerir de la presencia del robot, lo que hace que, mientras tanto, éste pueda ser usado en otro lugar, eliminando así el riesgo por interactuar con él.

Al ser un lenguaje textual, resulta más fácil documentar y modificar los procedimientos a realizar por el robot, además de permitir la simulación del comportamiento programado. En contrapartida, requiere de más complejidad y tiempo a la hora de describir trayectorias complejas, además de requerir que el operario posea ciertos conocimientos técnicos, disponer de antemano de las coordenadas de las posiciones de interés del entorno de trabajo y de poderse producir problemas por una calibración incorrecta.

2.3.3. Programación híbrida

Puesto que la mayoría de ventajas e inconvenientes de los tipos de programación anterior son contrarios entre sí, el hecho de combinarlos hace que se eliminen parte de los inconvenientes.

En la programación híbrida, se edita un programa textual donde las instrucciones van referidas a posiciones simbólicas, que se consiguen mediante el método de guiado, por lo que se evitan los problemas por calibración y las coordenadas de interés se obtienen de una forma más fácil.

2.4. Estudio de mercado

Teniendo en cuenta la fuerte presencia de los robots industriales en el mercado por sus diversas configuraciones y la versatilidad que presentan cada una de ellas, el objetivo del presente trabajo es crear una aplicación que imite los tipos de programación descritos anteriormente, mejorando así las aplicaciones que se le pueden dar al producto que se presentará en el siguiente subapartado.

En la actualidad, la realización de proyectos que incluyan el montaje de un brazo robótico y el movimiento de éste a través de una aplicación para tablets o móviles es muy extendida. A pesar de ésto, el uso que se les da está orientado únicamente al movimiento de cada articulación y a realizar estos movimientos, previamente definidos, de forma consecutiva. Aquellos proyectos que vayan más allá de este objetivo, suelen ser productos acabados que se venden en el mercado como kits de montaje con software incluido, tanto para el microcontrolador como para el dispositivo desde donde se controla, que puede ser un mando, un dispositivo móvil, tablet o una aplicación de ordenador. En este trabajo, se pretende crear algo así, pero con un software en el microcontrolador que permita el uso del robot desde cualquier tipo de aplicación, teniendo conocimiento de las órdenes predefinidas y de cómo crear nuevas.

En la página web de Arduino dedicada a los proyectos con placas de desarrollo Arduino (http://playground.arduino.cc/ o https://create.arduino.cc/projecthub), se pueden encontrar proyectos sobre la construcción de brazos robóticos, de los cuales, aquellos controlados desde otro dispositivo, suelen usar aplicaciones prehechas, es decir, aquellas que sirven para mover el robot y que ya contienen un protocolo de comunicación que se ha de adaptar en el proyecto o que son hechas por el fabricante del robot. En el caso de programar el microcontrolador, muchos de los proyectos lo hacen a través de aplicaciones o entornos de programación que permiten una programación más sencilla (programación visual) para aquellas personas con pocas nociones de programación, tales como Snap4Arduino o Scratch for Arduino (S4A), las cuales son versiones de Snap! y Scratch, respectivamente, orientadas a la programación de los microcontroladores de las placas de desarrollo Arduino.

A continuación, se procede a explicar algunos de los proyectos que ejemplifican lo anteriormente dicho.

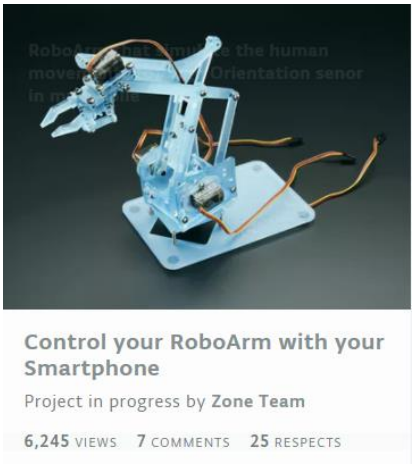


Ilustración 2.13. Portada de proyecto de control del robot MeArm desde un móvil [21]

En la Ilustración 2.13, se puede observar un proyecto orientado al movimiento del robot MeArm a través de un *Smartphone*, usando una placa de desarrollo Arduino/Genuino Uno y el Shield de Android 1Sheeld, el cual cuenta con una aplicación móvil que permite usar los componentes que incorpora cualquier Smartphone actual. En este caso, se hace uso del giroscopio y el sensor de proximidad de un móvil para mover el robot.

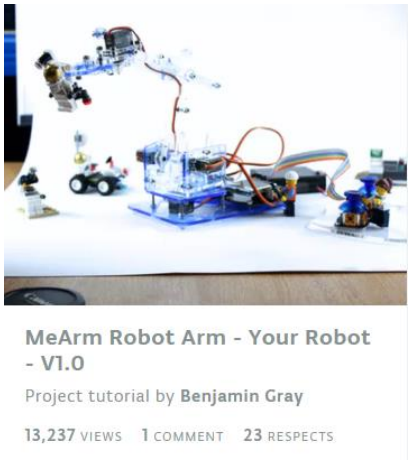


Ilustración 2.14. Ejemplo de proyecto usando el kit completo del robot MeArm [22]

En el ejemplo de proyecto de la Ilustración 2.14, se puede ver un proyecto parecido al anterior, pero, en esta ocasión, se usa el kit completo del robot MeArm, que incluye, además del robot, los cables y alguna herramienta, un mando con joystick, un controlador basado en Arduino y un porta pilas.

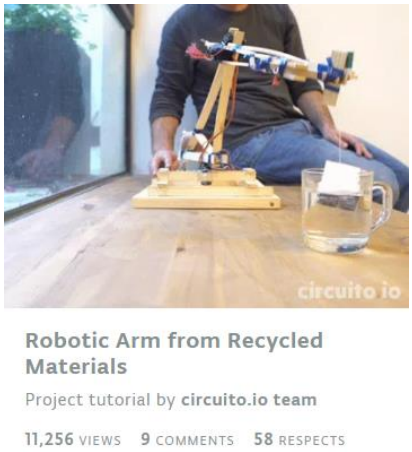


Ilustración 2.15. Ejemplo de proyecto de montaje y control de un robot hecho de materiales reciclados [23]

En la Ilustración de arriba se puede ver la portada de un proyecto de construcción de un brazo robótico a partir de materiales reciclados, una placa de desarrollo Arduino/Genuino Uno y algunos Shields de Arduino que permiten el control del robot desde un *joystick*.

En el siguiente proyecto de ejemplo, además de construir el brazo robótico, se usa la aplicación "Blynk", la cual permite la personalización de una interfaz de usuario con la que el usuario se comunicará con el microcontrolador, con la condición de que éste tenga conexión Wifi. Es por esto, que el autor usa una placa de desarrollo Arduino Yun. Adicionalmente a esto, usa el Robotic Toolkit de Matlab para implementar el cálculo cinemático inverso y además tener un modelo del brazo robótico en el ordenador.

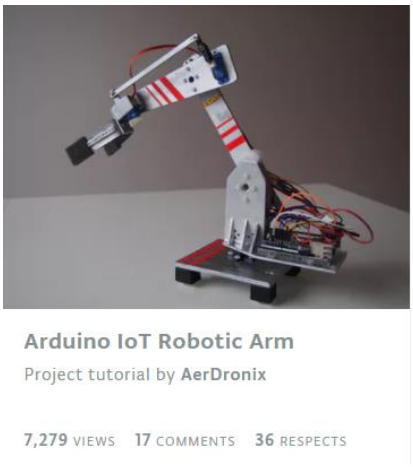


Ilustración 2.16. Ejemplo de proyecto de montaje y control de un robot con trayectoria [24]

Por último, en la Ilustración 2.17, se puede observar un kit Little Arm Original construido, el movimiento articular del cual se controla desde una aplicación hecha por los creadores del kit. Lo que es interesante de este robot es que está hecho a partir de piezas impresas, por lo que el creador vende los modelos 3D, piezas sueltas suficientes para su construcción y el kit completo, lo que hace que su precio no sea

tan alto y que se pueda usar de muchas maneras por sus piezas intercambiables. Puesto que cuenta con software para que se pueda controlar desde una consola en el ordenador y desde el móvil, no se requiere de nociones de electrónica o programación para sacarle partido a este kit.

Ilustración 2.17. Ejemplo de proyecto de control del kit Little Arm Original desde un móvil [25]

Otros ejemplos se pueden encontrar en blogs o en la plataforma de Youtube en forma de tutoriales o presentación del resultado del proyecto realizado.

Como se ha podido observar, existe toda una gama de kits parecidos al Tinkerkit Braccio Robot, con diferentes características y de diferentes precios. Cabe añadir que una alternativa a los kits, que puede ser algo más barata, es la adquisición de robots de ocio, los cuales, con un par de conocimientos de electrónica, se pueden personalizar para ser programables, como en el ejemplo de la Ilustración 2.18.

Ilustración 2.18. Robot Arm Edge [26]

Teniendo en cuenta todo lo descrito anteriormente, se podría decir que el presente trabajo añade a la premisa de controlar un robot desde alguna interfaz, flexibilidad en la comunicación, ya que se pretende hacer de la forma más genérica posible, y flexibilidad en la programación de movimientos, puesto que, en cierta manera, se creará un entorno de programación.

2.5. Tinkerkit Braccio Robot

En este trabajo se hará uso del Tinkerkit Braccio Robot de Arduino, un kit de montaje de un brazo robótico hecho de piezas de plástico y 6 servomotores, los cuales son controlados usando un Shield diseñado para encajar en una placa de desarrollo de Arduino y cuyo precio, actualmente, es de 199 € más gastos de envío.

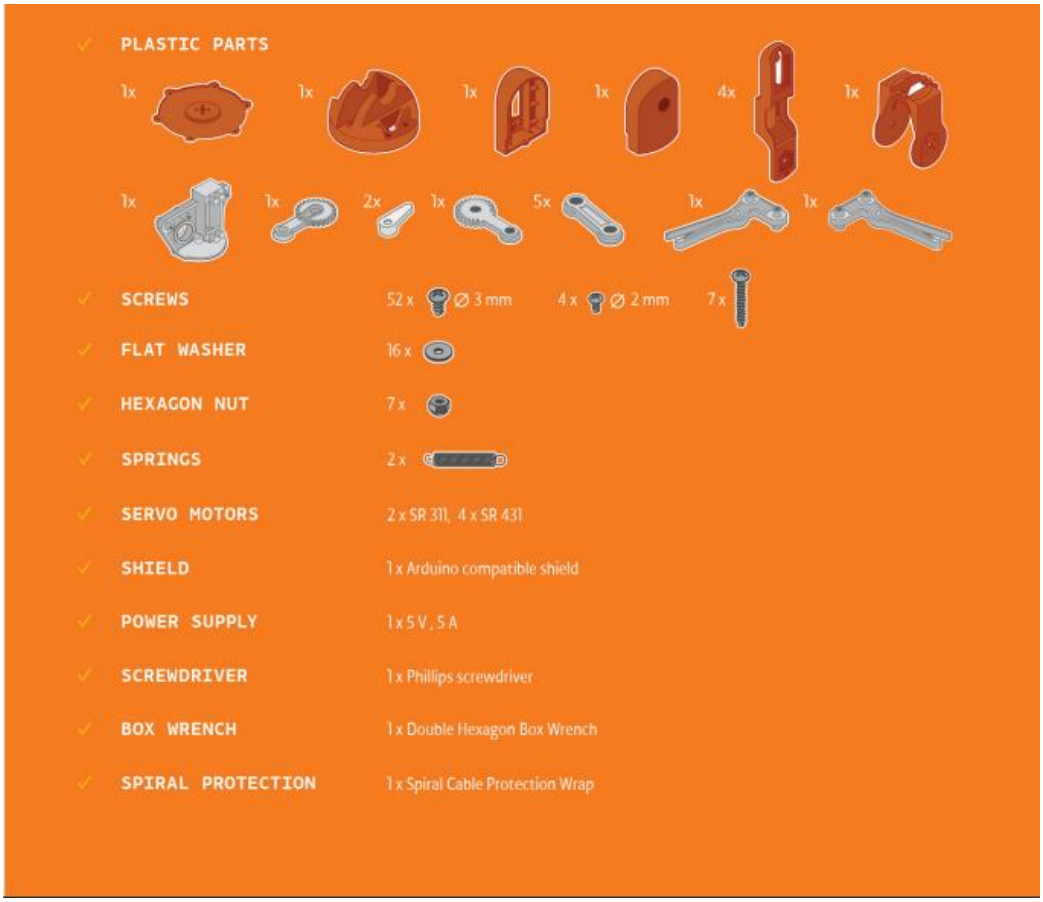


Ilustración 2.19. Contenido de la caja del Tinkerkit Braccio Robot según el manual de uso [27]

Ilustración 2.20. Contenido de la caja del Tinkerkit Braccio Robot [28]

2.5.1. Características

Una de sus características más notables es que su estructura no es única, por lo que es posible adaptarla a la aplicación que se le desee dar, tal y como se muestra en la Ilustración 2.21.

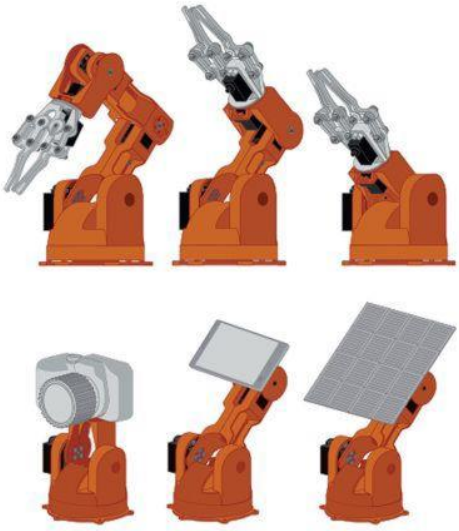


Ilustración 2.21. Posibles configuraciones estructurales de robot Braccio [29]

En lo que respecta a los detalles más técnicos, se resumen en las siguientes tablas, la cuales muestran primero las características más generales y, seguidamente, los detalles de los servomotores.

Tabla 2.4. Características técnicas del robot Braccio [1]

Característica	Detalles
Peso	0,792 kg
Distancia de funcionamiento máxima	80 cm
Máxima altura	52 cm
Anchura de la base	14 cm
Anchura de la pinza	9 cm
Longitud del cable	40 cm
Capacidad de carga	0,15 kg / 32 cm
Capacidad de carga en configuración mínima	0,4 kg
Fuente de alimentación	5 V, 5 A
Consumo del Shield	0,02 W
Corriente máxima deñ Shield	De M1 a M4: 1,1 A De M5 a M6: 0,75 A
Servomotores	2 x SR 311 + 4 x SR 431
Medidas del Shield PCB	6,858 x 5,334 cm

Tabla 2.5. Características técnicas de los servomotores SR 311 y SR 431 – Dual Output Servo [1]

Servomotores SR 311		**Servomotores SR 431 – Dual**	
Característica	**Detalles**	**Característica**	**Detalles**
Señal de control	PWM Analógica	Señal de control	PWM Analógica
Esfuerzo de torsión	4,8 V: 3,1 kg·cm 6,0 V: 3,8 kg·cm	Esfuerzo de torsión	4,8 V: 12,2 kg·cm 6,0 V: 14,5 kg·cm
Velocidad	4,8 V: 0,14 s/60° 6,0 V: 0,12 s/60°	Velocidad	4,8 V: 0,20 s/60° 6,0 V: 0,18 s/60°
Rango de rotación	180°	Rango de rotación	180°
Dimensiones	31,3 x 16,5 x 28,6 mm	Dimensiones	42,0 x 20,5 x 39,5 mm
Peso	0,027 kg	Peso	0,062 kg

2.5.2. Montaje

El kit incluye un manual de montaje paso a paso con unas ilustraciones muy precisas. A continuación, se presenta un reportaje fotográfico del proceso de montaje del kit usado.

Ilustración 2.22. Contenido del kit Tinkerkit Braccio Robot

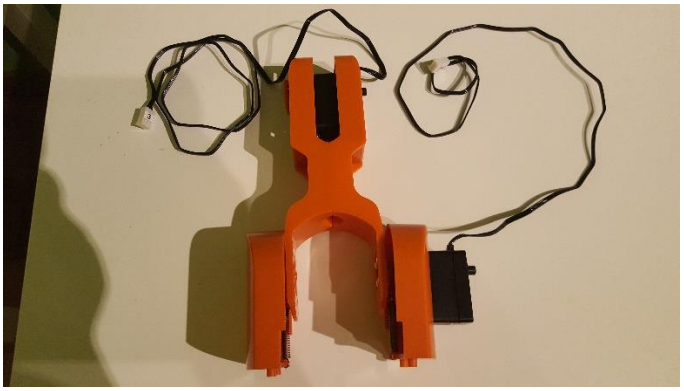


Ilustración 2.23. Montaje del hombro y el codo

Ilustración 2.24. Montaje de la base

Ilustración 2.25. Montaje de la muñeca de movimiento vertical

Ilustración 2.26. Montaje de la muñeca de movimiento rotacional

Ilustración 2.27. Montaje completo según el manual de montaje

Después de estar montado, al comparar el resultado final con el de la portada del manual (Ilustración 1), se distinguía una disposición de la base distinta. Al parecer, el paso 9 del manual contiene un error que, a menos que el usuario se percate al momento, se arrastra hasta el final del montaje. Esta disposición distinta no supone mayor problema, ya que funcionaría de la misma manera, pero con la base girada 180°. A pesar de esto, se ha optado por la corrección de este "defecto", que pasa por desmontar una pequeña parte de la base, y el resultado final es el siguiente:

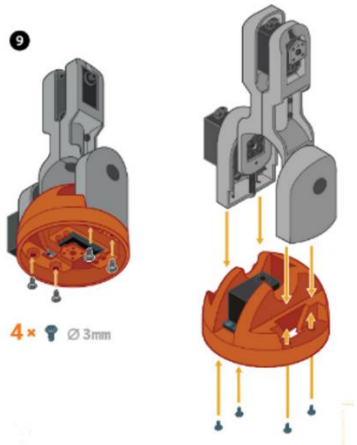


Ilustración 2.28. Captura del noveno paso del manual de montaje [27]

Para el control de este robot, una vez montado, Arduino proporciona una librería que incluye dos funciones básicas:

- Begin(): vincula las señales de control a los pines en los que se encuentran los servomotores y mueve el robot hacia una posición inicial. De esta manera, si no se posiciona como se espera, se han de calibrar los servomotores, lo cual se ha de hacer desmontado las piezas plásticas solidarias al eje de rotación de los servomotores y colocándolas en la posición en que deberían estar.

- ServoMovement(): permite mover todas las articulaciones del robot si se le proporciona como argumentos los ángulos a los que se desea mover cada articulación y un retardo entre cada ciclo de movimiento, que implica mover un grado cada articulación (aquellas que se hayan de mover). Los movimientos se han de realizar completamente, ya que el ciclo de movimiento se encuentra dentro de un bucle, lo que imposibilita realizar otras acciones como cálculos o recibir o enviar órdenes desde y hacia el microcontrolador mientras se produce el movimiento.

Ciertamente, con estas dos instrucciones es suficiente para sacarle algo de provecho a este kit, ya que basta con encadenar instrucciones *Servomovement*, pero el mayor inconveniente de este método es saber a priori cuáles serán los ángulos de destino de las articulaciones, además del hecho de haber de hacer movimientos completos, sin la posibilidad de usar la Arduino entre tanto, para calcular o realizar otra acción.

Por esta razón, se modificará la estructura de esta librería y se configurará de manera que las instrucciones de movimiento sean más flexibles y permitan el uso del microcontrolador a la vez que el robot se mueve.

3. Funcionalidad y especificaciones

Como se ha comentado anteriormente, el objetivo de este trabajo es proporcionar un uso del robot Braccio similar al de un robot manipulador industrial, por lo que se han de establecer similitudes y diferencias entre ellos para, al final, definir cómo ha de ser el algoritmo que permita la programación del robot Braccio.

La programación por guiado pasivo no es posible, ya que al desconectar los servomotores el robot Braccio, puesto que éstos incorporan los sensores de posición que en un robot industrial estarían separados, no se podría aprender las posiciones articulares deseadas. Por otro lado, la opción del guiado pasivo por maniquí es posible, pero se convertiría en otro tipo de trabajo en el que se habría de diseñar e implementar ese maniquí. El último tipo de guiado pasivo, en el que el robot se mueve solidario a cómo lo mueva el usuario, también es posible, pero de la misma manera que con el maniquí, se convertiría en otro tipo de trabajo, al haber de incorporar los sensores e implementar la lógica que permita este tipo de programación.

Por todo lo dicho y con la intención de ahorrar costes, la programación por guiado activo es la mejor opción dentro la programación por guiado. Es por esto, que el algoritmo ha de permitir guardar en memoria posiciones articulares actuales del robot y darles un nombre.

Por otro lado, es incuestionable que la programación textual es necesaria, por lo que, finalmente, se concluye que se realizará un algoritmo que permita una programación híbrida.

Este algoritmo requiere de alguna interfaz para que el usuario pueda interactuar con el robot, la cual, en este caso, se trata de una aplicación móvil y de la misma consola del entorno de programación de Arduino. El medio por el cual se comunicarán será por Bluetooth, al ser la opción de menos coste, y por comunicación serie, respectivamente.

Por lo tanto, las especificaciones que ha de cumplir el algoritmo introducido en el microcontrolador son que ha de poder permitir el movimiento del robot, establecer comunicaciones por las cuales enviar y recibir datos, interpretar los datos recibidos y que, dentro de esta interpretación, se puedan ejecutar órdenes de movimiento articular del robot, de movimiento referenciado a una posición previamente guardada y de manipulación de posiciones de interés y de programas creados, es decir, que éstos se puedan guardar, modificar y eliminar.

A continuación, se procede a definir cómo debería ser la aplicación móvil que se implementará para comprobar el correcto funcionamiento del algoritmo creado.

Para empezar, se debe crear una entidad que inicialice y gestione todo aquello necesario para crear la comunicación por Bluetooth con el robot. Así mismo, se requiere de una pantalla inicial desde la cual el usuario establezca esta comunicación.

Una vez estén comunicados el móvil y el microcontrolador, la siguiente pantalla debe de mostrar los botones que representen los modos de operación disponibles, que serán el "Jogging/Teach Pendant" y el modo "Program", los cuales tendrán, cada uno, su propia pantalla.

El modo "Jogging/Teach Pendant" habría de permitir el movimiento del robot mediante botones que aumenten o disminuyan el valor de los ángulos de cada articulación ("Jogging, Joint mode") o el valor de las coordenadas XYZ ("Jogging, XYZ mode") del sistema de coordenadas fijo en la base del robot. Puesto que se ha elegido un tipo de programación híbrida donde las posiciones articulares actuales se puedan guardar, sería necesaria una pantalla de gestión de éstas. Con esto dicho, se pueden diferenciar tres pantallas distintas dentro del modo "Jogging/Teach Pendant":

- "Jogging, Joint mode"
- "Jogging, XYZ mode"
- Gestión de posiciones articulares guardadas

En principio este trabajo debe servir para acercar a un público con pocos conocimientos técnicos a la programación de robots industriales, por lo que las pantallas deben de facilitar la programación en la mayor medida que se pueda. En este contexto se presentan dos propuestas, en lo referido a la pantalla del modo "Program". La primera es la opción de usar un texto plano que contenga instrucciones predefinidas, de las cuales se verifique su correcta sintaxis y se codifiquen según el set de instrucciones programadas en el microcontrolador. La segunda opción es crear una lista dinámica en la que, en cada elemento, aparezca un menú desplegable con la lista de comandos disponibles y, según el que se escoja, aparezcan otros menús desplegables con los argumentos que requiera ese comando, a la derecha del primero, que contengan las opciones disponibles. Por otra parte, sería interesante poder guardar y cargar los programas creados, junto con las posiciones articulares guardadas en ese momento en algún archivo local del móvil. Para el presente trabajo, se escogerá la primera opción y, la segunda se realizará en caso de disponer del tiempo suficiente para su implementación.

Para cumplir el objetivo principal del trabajo, las instrucciones básicas que debe poder soportar el modo de programación son:

Tabla 3.1. Instrucciones que se implementarán en el modo de programación de la interfaz de usuario

<table>
<tr><th>Comando</th><th>Argumentos</th><th>Comentarios</th></tr>
<tr><td>HOP</td><td>Ninguno</td><td>Abre la pinza.</td></tr>
<tr><td>HCL</td><td>Ninguno</td><td>Cierra la pinza.</td></tr>
<tr><td>MOV Mmotor ángulo</td><td>1er argumento: motor que se desea mover.<table><tr><td>M1</td><td>M2</td></tr><tr><td>M3</td><td>M4</td></tr><tr><td>M5</td><td>M6</td></tr></table>2do argumento: ángulo de destino con rango según el motor que se seleccione.</td><td>Orden de movimiento de una articulación (motor) en concreto hacia el ángulo de destino indicado.</td></tr>
<tr><td>MOV coordenada valor</td><td>1er argumento: coordenada X, Y o Z que se desea modificar.

2do argumento: valor de la coordenada a la que se desea llevar el elemento terminal (la pinza).</td><td>Orden de movimiento de todas las articulaciones para alcanzar el valor de la coordenada modificada.</td></tr>
<tr><td>MOV Pvalor</td><td>Posición articular guardada previamente, con un índice que la identifica.</td><td>Orden de movimiento de todas las articulaciones hacia los ángulos especificados por la posición articular indicada.</td></tr>
</table>

4. Diseño electrónico

En esta sección se presentan los materiales usados, aparte de kit, junto a los motivos que han llevado a su elección.

4.1. Selección de componentes

El kit de montaje usado no incluye el microcontrolador que ha de gestionar el movimiento del robot, por lo que se ha de elegir uno que se ajuste a la aplicación que se le dará en este trabajo.

Por otra parte, existe una necesidad de poder comunicar el microcontrolador con el móvil. Las dos opciones a valorar son el uso de un módulo de WIFI o un módulo de Bluetooth. Teniendo en cuenta que se pretende abaratar costes y que no se requiere de la posibilidad de usar el robot de forma muy remota, ya que se ha de evitar el uso del robot sin vigilar que no se estropee por una mala operación, la opción que parece ajustarse mejor a las necesidades de este trabajo es el uso de un módulo de Bluetooth.

4.1.1. Microcontrolador

Un microcontrolador es un sistema electrónico digital programable compuesto de una unidad central de procesamiento o CPU, una memoria y unidades de entrada y salida.

Inicialmente, se han planteado como opciones las placas de desarrollo de las marcas Arduino y Raspberry, ya que el uso de éstas en la construcción de prototipos es muy extendido, por la flexibilidad que ofrecen para ampliar y modificar montajes, la cantidad de información que se puede conseguir de ellas, gracias a la amplia comunidad de usuarios y su flexibilidad en la programación, ya que se pueden compilar y ejecutar desde casi cualquier entorno de programación. En el caso de Arduino, su mayor ventaja es la facilidad que presenta de crear aplicaciones con sus librerías propias, sin requerir de rigurosas nociones de programación, y en el caso de Raspberry, las características técnicas, muy superiores a las de Arduino, que ofrece a bajo coste.

La aplicación a realizar no requiere de un pinaje numeroso ni de una gran capacidad de memoria del microcontrolador, por lo que la opción de menor coste y más apropiada es el microcontrolador Arduino UNO Rev3. Otro de los motivos de su elección es que el Shield que alimenta a los servomotores del robot Braccio está hecho para usarse con una placa de desarrollo de Arduino, aunque esto no es obligatorio, ya que se puede adaptar para trabajar con Raspberry.

El microcontrolador ATmega328P de 8 bits, en el que está basada la placa de desarrollo Arduino Uno, presenta las siguientes características:

- Alto rendimiento.
- Bajo consumo con 5 modos de ahorro de consumo seleccionables por software.
- 32 KB de memoria de programa, tipo Flash, de los cuales 0,5 KB son destinados al *bootloader*
- 2 KB de memoria SRAM
- 1 KB de memoria EEPROM
- Velocidad de la CPU de 20 MPIS (millones de instrucciones por segundo)
- 23 pines de entrada/salida
- 32 registros de propósito general
- 2 temporizadores/contadores de 8 bits y 1 de 16 bits habilitados con modo de comparación
- Interrupciones internas y externas
- Periféricos de comunicación digital: USART ("Universal Synchronous and Asynchronous Receiver-Transmitter") programable, 2 puertos SPI ("Serial Peripheral Interface") y 1 bus I2C
- Un conversor analógico digital de 10 bits y 6 canales
- Otros periféricos: 6 PWM ("Phase Width Modulation"), 1 CCP ("Capture/Compare/PWM"), 1 "Input Capture"
- Rango temperatura de funcionamiento de -40 a 85 °C
- Rango de voltaje de funcionamiento de 1,8 a 5,5 V

Por otro lado, las características técnicas de la placa de desarrollo son las siguientes:

- Voltaje de funcionamiento de 5V
- Voltaje de entrada recomendado de 7 a 12 V
- Voltajes de entrada límites de 6 V y 20 V
- 14 pines de entrada/salida digital, 6 de los cuales proporcionan una salida PWM
- 6 pines de entrada analógica
- Corriente continua por los pines de entrada/salida de 20 mA
- Corriente continua por el pin de 3,3 V de 50 nA
- Velocidad de reloj de 16 MHz
- Proporciona un LED montado en la placa, que se puede controlar desde el pin digital 13.
- Dimensiones: 6,86 x 5,34 cm
- Peso: 25 g

En la siguiente ilustración, se presentan la estructura de la placa de desarrollo Arduino Uno Rev3.

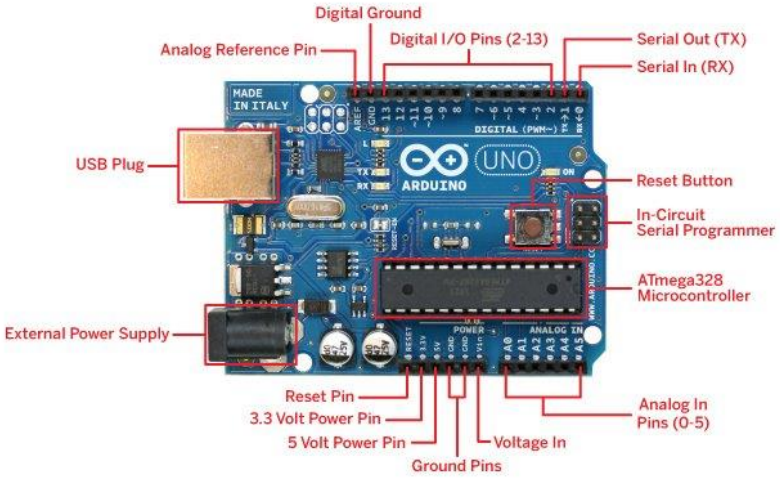


Ilustración 4.1. Estructura de la placa de desarrollo Arduino Uno Rev3 [32]

4.1.2. Módulo de Bluetooth

Bluetooth se refiere a un protocolo estandarizado de comunicación inalámbrica de corto alcance (<100m), bajo coste y bajo consumo que permite enviar y recibir datos de forma segura a 2,4 GHz.

Los módulos de Bluetooth pueden funcionar de dos maneras, como maestros, los cuales pueden estar conectados a hasta 7 esclavos, sobre los cuales arbitra la transferencia de datos o, como esclavo, pudiendo éste sólo estar conectado a un maestro. La red que forman se denomina piconet.

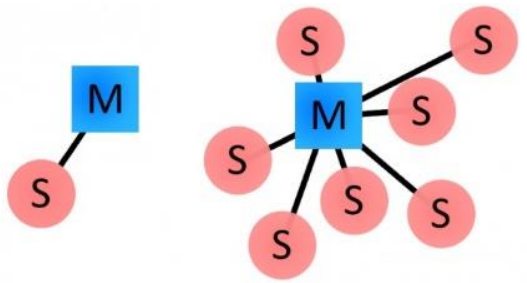


Ilustración 4.2. Tipos de piconet [33]

Cada módulo se identifica por una dirección única de 48 bits y un nombre, que puede estar por defecto o se puede modificar al que se desee. La comunicación entre dos dispositivos con Bluetooth se realiza mediante la introducción de un PIN o clave de acceso, durante un procedimiento llamado emparejamiento, a partir del cual, los dos dispositivos se pueden conectar directamente las siguientes veces, sin intervención manual, mientras se encuentren cerca entre ellos.

Los módulos de Bluetooth más usados en el mercado, por su bajo coste y su facilidad de uso, son los HC-05 y los HC-06. Aunque el hardware y el precio es el mismo, la diferencia entre estos dos módulos es que, el primero, puede trabajar como maestro o esclavo y, el segundo, únicamente como esclavo.

Por esta razón, el módulo HC-05 es más flexible en cuanto al uso que se le puede dar, pero a su vez es más complicado de configurar y usar.

Puesto que la aplicación que se le dará en este trabajo es para establecer una comunicación entre dos dispositivos, donde el maestro es el módulo de Bluetooth del móvil, el módulo HC-06 parece ser la elección más idónea.

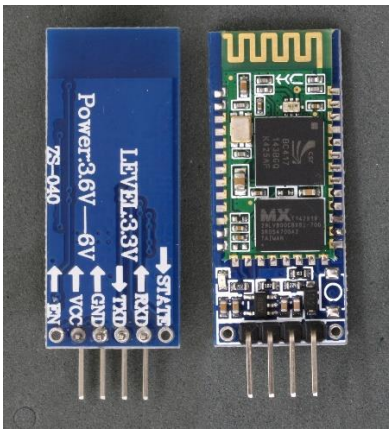


Ilustración 4.3. Módulo de Bluetooth HC-06 ZS-040 [34]

Como se puede observar en la ilustración anterior, el módulo se compone de 4 pines, 2 de ellos son destinados a la alimentación (VCC y GND) y los otros 2, a la recepción (RXD) y transmisión de datos (TXD).

Para su configuración, se usan los comandos de atención o AT (se envía una orden del tipo AT+orden) que se listan a continuación:

Tabla 4.1. Comandos AT disponibles

<table>
<tr><th>Comando</th><th>Respuesta</th><th>Uso</th></tr>
<tr><td>AT</td><td>OK</td><td>Test de funcionamiento, mientras no esté emparejado.</td></tr>
<tr><td>AT+BAUD<p></td><td>OK<r></td><td>Configuración del “Baud rate”, que por defecto es de 9600 bps, según la siguiente tabla:

Tabla 4.2. Comandos AT [35]
<table><tr><th><p></th><th><r></th><th>Remarks</th></tr><tr><td>1</td><td>1200</td><td>set to 1200bps</td></tr><tr><td>2</td><td>2400</td><td>set to 2400bps</td></tr><tr><td>3</td><td>4800</td><td>set to 4800bps</td></tr><tr><td>4</td><td>9600</td><td>set to 9600bps (Default)</td></tr><tr><td>5</td><td>19200</td><td>set to 19200bps</td></tr><tr><td>6</td><td>38400</td><td>set to 38400bps</td></tr><tr><td>7</td><td>57600</td><td>set to 57600bps</td></tr><tr><td>8</td><td>115200</td><td>set to 115200bps</td></tr><tr><td>9</td><td>230400</td><td>set to 230400bps</td></tr><tr><td>A</td><td>460800</td><td>set to 460800bps</td></tr><tr><td>B</td><td>921600</td><td>set to 921600bps</td></tr><tr><td>C</td><td>1382400</td><td>set to 1382400bps</td></tr></table></td></tr>
<tr><td>AT+NAME<nombre></td><td>OK<nombre></td><td>Configuración del nombre del módulo HC-06. Por defecto se muestra como HC-06.</td></tr>
<tr><td>AT+PIN<nnnn></td><td>OK<nnnn></td><td>Configuración de la clave de acceso para el correcto emparejamiento con otros módulos de Bluetooth, que por defecto es 1234.</td></tr>
<tr><td>AT+PN</td><td>OK NONE</td><td>Establece no comprobar la paridad, la cual es por defecto</td></tr>
<tr><td>AT+PO</td><td>OK ODD</td><td>Establece la comprobación por paridad impar.</td></tr>
<tr><td>AT+PE</td><td>OK EVEN</td><td>Establece la comprobación por paridad par.</td></tr>
<tr><td>AT+VERSION</td><td>OK<Repuesta></td><td>Comando para saber la versión</td></tr>
</table>

Las características técnicas más relevantes del módulo de Bluetooth HC-06 ZS-040 V1.8, son las siguientes:

- Corriente durante el emparejamiento: 30 a 40 mA
- Corriente durante la comunicación: 8 mA
- Voltaje de funcionamiento: 3,3 a 6 Vdc
- Dimensiones: 4,3 x 1,5 x 0,7 cm
- Alcance: 10 metros
- Incluye una antena PCB de 2,4 Ghz
- Interface vía UART
- Rango de temperatura de funcionamiento: -25 a +75 °C

4.2. Esquema de conexión

Si se observa el pinaje del Shield proporcionado en el kit, se puede comprobar que encaja con el pinaje de la placa Arduino UNO Rev3. Por esta razón, en la ilustración X se muestra el esquemático sin el Shield encajado.

Los pines de la placa de desarrollo asociados a los motores M1, M2, M3, M4, M5 y M6 son los pines digitales 11, 10, 9, 6, 5 y 3, respectivamente. Puesto que estos pines ya están ocupados y los pines digitales 1 y 2 son exclusivos de la comunicación serie con la consola del IDE de Arduino, la cual se pretende aprovechar, los primeros pines digitales libres para establecer la comunicación serie con el módule de Bluetooth HC-06 son el 2 y el 4. Respecto a la alimentación del módulo, el pin Vcc va conectado a al pin de 5 V de la placa y el GND a uno de los dos disponibles en la Arduino UNO.

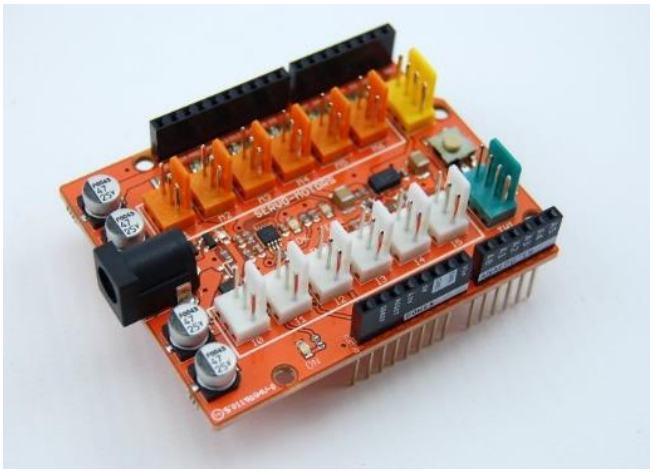


Ilustración 4.4. Arduino Shield para Tinkerkit Braccio Robot [37]

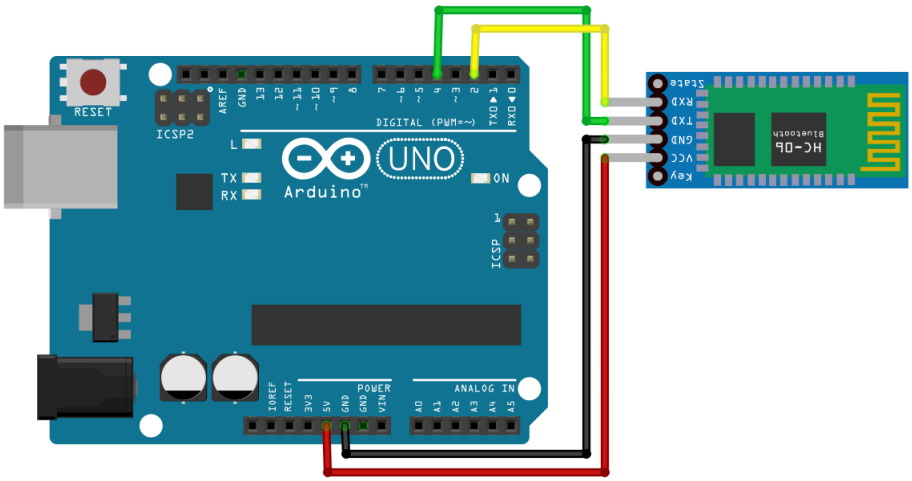


Ilustración 4.5. Esquema de conexión

5. Programación del robot Braccio

Arduino proprociona un entorno de desarrollo integrado (IDE) de código abierto, cuyo lenguaje es una variante de C.

El archivo principal y base es de extensión .ino, el cual se compone de las siguientes partes:

- Uso de directivas tales como "#include", para adjuntar archivos externos al que se está editando, o "#define" para declarar valores predefinidos.
- Declaración de variables y funciones, las cuales serán globales.
- void setup(): Esta función se ejecuta una sola vez al principio del programa y es aquí donde se han de inicializar las variables creadas anteriormente. Se ha de tener en cuenta que cualquier variable creada dentro de la función, será local, por lo que el resto de programa no podría acceder a ellas.
- void loop(): Se trata de un bucle infinito, en el que se ha de implementar el código que describe el comportamiento que tendrá el microcontrolador, teniendo en cuenta que éste se repetirá de forma infinita.
- Implementación de las funciones declaradas anteriormente.

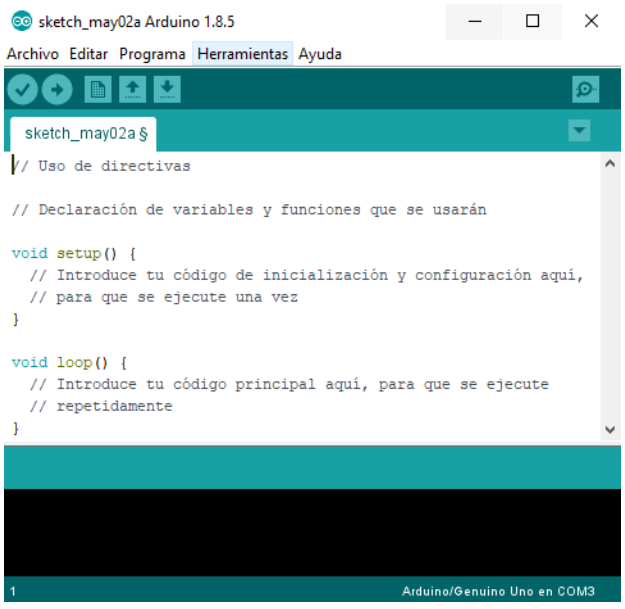


Ilustración 5.1. Interfaz del entorno de desarrollo integrado de Arduino

En la entorno del IDE se compone de menús desplegables (arriba a la izquierda), una pequeña consola que muestra los mensajes relativos a la compilación y ejecución del código (abajo), el cuadro de texto en el que se introduce el código a ejecutar y seis botones con las funciones de compilar y ejecutar el código creado, abrir una nueva ventana, abrir un archivo, guardar un archivo modificado, abrir la

consola de comunicación con el microcontrolador, y abrir un desplegable con los archivos de los que se compone el proyecto, respectivamente y en orden de izquierda a derecha.

El uso del entorno de Arduino es muy extendido por la facilidad que ofrece a la hora de implementar programas básicos de control de hardware, ya que proporciona un gran número de librerías que, al ser de código abierto, se van creando y mejorando continuamente, gracias a la gran comunidad que ha conseguido. El objetivo de esto es que cualquier persona pueda llevar a cabo un proyecto de electrónica sin tener apenas conocimientos sobre ello. La contrapartida que presenta es que si se pretenden objetivos muy técnicos dentro del proyecto, el uso de estas librerías deja de ser recomendable por la cantidad de recursos que usa, teniendo en cuenta que las características técnicas de las placas Arduino son limitadas.

Otro detalle a comentar es la simplicidad del editor de texto del IDE de Arduino, el cual dificulta en cierta manera la programación. Por esta razón, se ha optado por el uso de otros editores de texto, como Visual Studio, que facilitan en gran medida la implementación de código, gracias al texto predictivo, la señalización de parámetros con colores y el autocompletado a la hora de crear funciones. Puesto que el complemento de Visual Studio para Arduino es algo complicado de configurar, para la compilación y ejecución del código se optará por el uso del IDE de Arduino.

Por último, con el objetivo de estructurar el código y no empaquetarlo en un mismo archivo, se hará el uso de diversas librerías propias u archivos externos al archivo base de extensión .ino, programados en C++ adaptado a Arduino, es decir, usando lenguaje orientado a objetos, cuyo desarrollo se explicará en los siguientes apartados.

5.1. Desarrollo del algoritmo

La primera cosa que ha de poder hacer el algoritmo que se desarrollará es mover el brazo articulado Braccio, cosa que se hará mediante el uso de la librería “Braccio.h” de código abierto que se proporciona en el perfil de Github (programa de control de versiones) de Anduino, para cuyo uso basta con abrir uno de los ejemplos que se proporcionan en la misma página web. En general, la estructura de estos ejemplos pasa por crear un objeto global tipo Braccio, inicializarlo desde el “setup” y, si se desea un solo ciclo de movimiento, usar la función de movimiento del brazo dentro de esta misma función. De otra forma, se habrá de implementar dentro de la función “loop”.

En la librería “Braccio.h” se encuentra la inicialización de los servomotores, lo cual se traduce en que se declaran los pines de control, que corresponden a los pines digitales 11, 10, 9, 6, 5, y 3, y se establecen los ángulos iniciales de cada articulación. Con el objetivo de evitar que el robot se dañe a causa de un movimiento brusco hacia los ángulos iniciales, se proporciona una función llamada “SoftStart” o encendido suave que implementa dos señales PWM en el pin digital 12. Puesto que no se

disponen de las características técnicas del Shield, debido al cambio de propietarios de la compañía Arduino, cabe suponer que dicho pin 12 influirá en el comportamiento de los servomotores. Para acabar, la última función que se incluye es la denominada “ServoMovement”, la cual, a partir de los argumentos de entrada, que corresponden a los ángulos de cada articulación y el retardo entre cada movimiento de 1° por articulación, establece los ángulos de destino y entra en un bucle que no acaba hasta completar el movimiento hacia la posición articular especificada.

A partir de este punto, se ha planteado la modificación de dicha librería a causa de su baja flexibilidad para ser integrado en un algoritmo de más complejidad, como el que se pretende realizar, cosa que se explicará en más detalle más adelante.

El siguiente paso a trabajar, es la obtención y procesamiento de datos a partir de algún medio de comunicación, que para empezar, puede ser la misma consola del IDE de Arduino. Para su realización, se lee cada uno de los caracteres que se introducen hasta el fin de línea, con la ayuda de las funciones incluidas en el objeto “Serial” propio de Arduino. Puesto que se ha establecido trabajar con comandos codificados, pero flexibles en cuanto a los argumentos y número de carácter que pueden contener, se ha requerido de la obtención de un archivo que permita crear listas dinámicas, denominada en este trabajo como “List.h”.

En la comunicación con elementos externos, la información llega en forma de caracteres que, para facilitar su tratamiento, se convertirán en números enteros.

Ilustración 5.2. Pantalla de vinculación de la aplicación para móvil “BlueTerm” [38]

Una vez se ha podido mover el robot desde la consola a partir de comandos codificados, se ha de empezar a trabajar en la comunicación con el módulo de Bluetooth HC-06 y, consiguientemente, con el móvil. Con el objetivo de poder enviar y recibir datos sin necesidad de desarrollar la aplicación móvil, se ha hecho uso de la aplicación “BlueTerm”, la cual se puede encontrar en la tienda “Google Play” y

que se trata de un pequeño programa que permite la vinculación con un dispositivo Bluetooth y el envío y recepción de datos. El módulo HC-06 se comunica con el microcontrolador por medio de una comunicación serie, por lo que su comportamiento es igual que el de la consola del IDE de Arduino, con la diferencia de que su declaración y configuración se ha de realizar de forma manual, es decir, mediante el uso de la librería "SoftwareSerial.h", se han de designar los pines de la placa a los cuales de conectará y se ha de realizar la gestión pertinente de su estado (lo que significa saber si se encuentra vinculado a algún dispositivo) y tratamiento de datos.

Para realizar la primera vinculación del módulo HC-06 con el móvil, se ha de activar la función de Bluetooth de éste último, de manera que, si el módulo se encuentra operativo, se visualizará su nombre (HC-06) en la lista de dispositivos encontrados. Al seleccionarlo, se pide el pin de acceso, que por defecto es 1234.

A continuación, una vez se han completado los objetivos anteriores, es el momento de pensar en las mejoras que se le pueden hacer a la librería "Braccio.h". Para empezar, es necesaria la posibilidad de establecer el ángulo de destino de una sola articulación, de manera que no se requiera saber la información del resto de articulaciones para poder mover el robot. Seguidamente, se ha de eliminar el bucle que no permite realizar ninguna otra operación mientras el robot se mueva, cosa que se implementará aprovechando el bucle infinito del archivo base ("Inicio.ino"), de manera que, por cada vuelta, se actualice la posición de los servomotores en un 1°, en el caso de que la posición de destino no sea igual a la actual. Por último, se le ha añadido la posibilidad de consultar las posiciones angulares actuales y la de configurar el tiempo de retardo que se ha de asegurar entre cada movimiento de 1° de cada articulación.

Con todo lo explicado anteriormente, se pueden reconocer tres subsistemas base: el de comunicación, que gestiona el envío y la recepción de datos y el estado del medio de comunicación; el de gestión de comandos referidos a los servomotores del brazo articulado; y, por último, el subsistema de procesamiento de datos recibidos.

Una vez conseguida la gestión de la comunicación serie con la consola del IDE de Arduino, a la vez que la de la información proveniente del móvil a través del módulo HC-06, el siguiente paso a tratar es la implementación de los comandos que usará la aplicación móvil, con la finalidad de cumplir las funcionalidades descritas para este trabajo. Dichos comandos, se presentarán en el siguiente subapartado y han ido surgiendo a medida que se iba desarrollando la aplicación de Android.

Finalmente, si todo lo anterior se comporta de forma correcta, se puede decir que se ha cumplido el objetivo marcado.

5.2. Protocolo de comunicaciones

La creación de los comandos actuales ha sido muy influenciada por el comportamiento de la aplicación móvil que se estaba desarrollando de forma paralela. A pesar de esto, se han pensado de manera que se puedan usar desde cualquier interfaz de usuario.

Cada comando consiste en una serie de uno a nueve números, separados por comas entre ellos, terminada con el carácter '#', que señala fin de línea y que ha sido necesario de aplicar porque, aunque las instrucciones se envían una a una desde el móvil, si se hace de forma rápida, se concatenan entre ellas en la cola de espera del módulo HC-06, lo cual conduce a su incorrecta interpretación.

Los comandos de control de movimiento articular han sido los primeros en implementarse, ya que de ellos dependen una gran cantidad de funcionalidades, que se verán a continuación, y, que corresponden al control manual del movimiento por articulación y por eje de coordenadas ("Joggging") y a las órdenes del control automático de movimiento por articulación, por eje o por posición articular, que implican la finalización de cada movimiento.

Con el fin de gestionar la vinculación entre el módulo HC-06 con el móvil, se han creado comandos de conexión y desconexión.

Seguidamente, para visualizar las posiciones articulares actuales desde el móvil, se ha añadido el comando "request", el cual, la ser invocado, devuelve un conjunto de seis números, donde cada elemento representa la posición articular de la articulación correspondiente al índice que ocupa en dicho conjunto.

A partir de la implementación del modo programación en la aplicación móvil, fue necesaria la introducción de comandos referidos a la apertura y cierre de la pinza, al paro de la ejecución del programa actual y a la gestión de posiciones, de manera que si se modifican en el móvil, también lo hagan en la memoria del microcontrolador. Estas modificaciones consisten en la creación, modificación y eliminación de posiciones articulares especificadas por el índice que ocupan en una lista de números enteros que las contiene.

Por último, se ha creído conveniente el uso de un comando que permitiera enviar texto entre medios de comunicación distintos, de manera que desde el móvil se pudiese enviar información que se pueda visualizar en la consola del IDE de programación y viceversa, es decir, desde la consola a la interfaz creada en el móvil.

A continuación, se procede a describir en detalle cada uno de los comandos que se han presentado a lo largo de esta sección.

Tabla 5.1. Descripción de las instrucciones implementadas

Número representativo	Nombre	Argumentos	Funcionalidad
0	"Connect"	1	Comando que indica el establecimiento de conexión con un elemento externo
1	"Disconnect"	1	Comando que indica que la conexión con un elemento externo se va a detener
2	"Request"	0	Comando que al ser invocado devuelve una lista de número enteros en la que cada uno de ellos representa la posición angular de la articulación correspondiente al índice que ocupa
3	"Sendto"	2	Comando que envía una cadena de texto hacia otro medio de comunicación.
4	"Jogging"	2	Comando que gestiona el control manual del movimiento del robot por articulación y por eje de coordenadas
5	"Move"	1 - 2	Comando que gestiona el control automático del movimiento del robot por articulación, por eje de coordenadas y por posición articular, previamente almacenada en memoria
6	"Hand"	1	Comando que gestiona la apertura y cierre total de la pinza.
7	"SavePos"	1 - 8	Comando usado para gestionar las posiciones articuladas
8	"Stop"	0	Comando usado para detener la ejecución del programa actual.

En los comandos 0, 1 y 3, el segundo argumento se refiere al tipo de comunicación sobre el que se desea realizar la acción y que puede ser:

Tabla 5.2. Tipos de comunicación disponibles e índice que les representa

Número representativo	Tipo de comunicación
0	Serial
1	Bluetooth

En el caso del comando 3, el tercer argumento se trata de la cadena de texto que se desea transmitir.

Respecto al comando 6, relativo al movimiento automático (programado) de la pinza, su segundo argumento denota la posición articular a la que se requiere llevar la pinza, como se muestra en la siguiente tabla:

Tabla 5.3. Codificación de la apertura o cierre total de la pinza

Número representativo	Posición articular
0	Pinza totalmente abierta a 10°
1	Pinza totalmente cerrada a 65°

El control manual del movimiento del robot requiere de dos argumentos adicionales, donde el primero denota la articulación o eje cuyo valor actual se desea incrementar (2), disminuir (1) o dejar igual (0), lo cual representa el último argumento.

Tabla 5.4. Segundo argumento del comando de movimiento automático o programado

Segundo argumento	
Número representativo	**Articulación o eje**
0	M1 o base
1	M2 u hombre
2	M3 o codo
3	M4 o muñeca (movimiento vertical)
4	M5 o muñeca (movimiento rotacional)

Segundo argumento	
Número representativo	**Articulación o eje**
5	M6 o pinza
6	X
7	Y
8	Z

En cuanto al comando de control automático de movimiento, su sintaxis puede estar formada por uno o dos argumentos adicionales, a causa de que existen tres modos:

- Control por articulación: presenta los mismos argumentos que en el caso del control manual, con la diferencia de que el tercer argumento se trata del valor del ángulo al que se desea llevar la articulación especificada (0 – 1 – 2 – 3 – 4 – 5).
- Control por eje de coordenadas: presenta los mismos argumentos que en el caso del control manual, con la diferencia de que el tercer argumento se trata del valor del eje especificado (6 – 7 – 8) al que se desea llevar el elemento terminal, en este caso, la pinza.
- Control por posición articular: en este caso, solo se requiere de un parámetro que indica el índice de la posición articular guardada a la que se desea llevar el robot.

Por último, la gestión de las posiciones articulares consiste en el guardado de una posición articular actual o especificada en un índice concreto y la eliminación de una posición concreta o de todas las existentes. La sintaxis de sus argumentos se muestra en la siguiente tabla:

Tabla 5.5. Argumentos del comando de gestión de posiciones articulares

Segundo argumento		Siguientes argumentos
Número representativo	Detalle	Detalle
0	Guardado de la posición articular actual	Índice de la posición a guardar
1	Guardado de una posición articular concreta	Índice de la posición a guardar seguido de la lista de ángulos que forman la posición articular
2	Borrado de todas las posiciones guardadas	-
3	Borrado de la posición articular especificada	Índice de la posición a eliminar

5.3. UML (Unified Modeling Language) Diagram

Un diagrama UML es una representación en la que se especifica, construye y documenta el modelado de un sistema de software en el lenguaje de programación orientado a objetos, incluyendo aspectos como las variables, funciones y relaciones entre clases.

Usando la versión simplificada de este método, en la Ilustración de a continuación se muestran los archivos que componen el código del microcontrolador y la relación entre ellos.

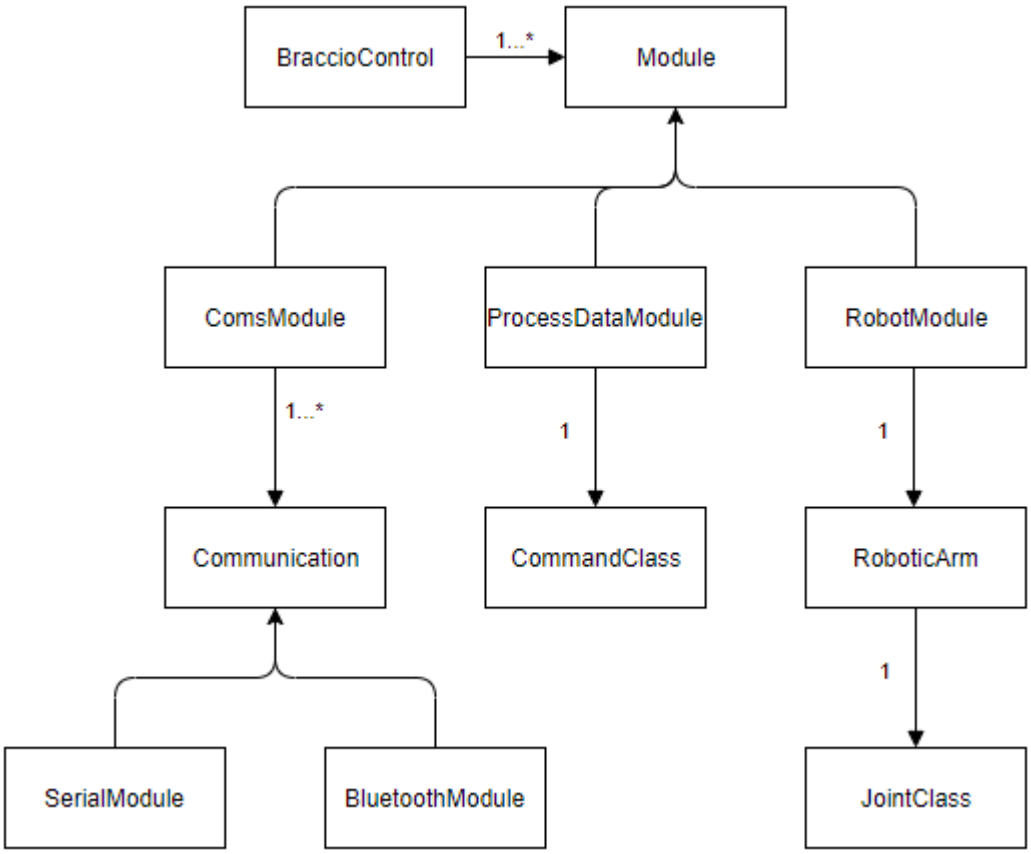


Ilustración 5.3. Diagrama UML de la aplicación de Arduino

5.4. Código del microcontrolador

El código completo introducido en el microcontrolador de la placa de desarrollo Arduino Uno Rev3 se encuentra adjunto en el Anexo A: Código implementado en Arduino. Cabe destacar que los comentarios se han realizado de inglés, de manera que pueda llegar a un público más amplio.

6. Diseño y programación de la aplicación para dispositivo móvil

En este apartado se presentarán los recursos utilizados para la implementación de la aplicación móvil que realizará la función de interfaz de usuario en este trabajo, el diseño de las pantallas según las especificaciones de funcionamiento y la relación entre ellas.

6.1. Diseño de la aplicación

De acuerdo a las especificaciones que ha de cumplir la aplicación móvil, ésta estará formada por seis pantallas:

1. Pantalla de vinculación con el robot
2. Menú principal
3. Pantalla de control manual del robot ("Jogging") y aprendizaje y gestión de posiciones de interés ("Teaching")
    a. Pestaña de control manual por articulación ("JOINT Jogging")
    b. Pestaña de control manual por coordenadas ("XYZ Jogging")
    c. Pestaña de gestión de las posiciones aprendidas
4. Pantalla de programación ("Program")

6.1.1. Pantalla de vinculación

Antes de poder utilizar la aplicación, se ha creído conveniente la vinculación previa obligatoria con el robot, ya que, de no ser así, las funciones de las demás pantallas no tendrían sentido.

En cuanto se entre a esta pantalla, se le pedirá al usuario que active el Bluetooth del móvil, mediante la ventana emergente de la Ilustración 6.1, en caso de que éste no lo esté. Mientras el usuario no encienda el Bluetooth, la aplicación no dejará de pedírselo.

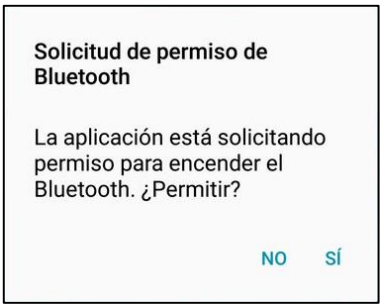


Ilustración 6.1. Ventana emergente de solicitud de uso del adaptador de Bluetooth del móvil

Ya con el Bluetooth activo, se podrá ver la lista “List of devices” que contiene elementos seleccionables con los nombres y direcciones de los dispositivos con los que el móvil de haya emparejado e intercambiado datos anteriormente.

Aprovechando que el primer emparejamiento es sencillo de hacer desde el móvil, no se cree conveniente la necesidad de implementar esta funcionalidad dentro de la aplicación. Por este motivo, si el módulo HC-06 no se encontrara en la lista, se habría de salir de la aplicación, realizar el emparejamiento, entrar de nuevo a la aplicación y seleccionar el botón “Search paired devices”, el cual refrescará la lista de dispositivos emparejados.

Al seleccionar uno de los elementos de la lista, se verificará si éste es el HC-06, suponiendo que este sea el único HC-06 disponible y que corresponda al módulo que permite el control del robot Braccio. En caso de no serlo, se le pedirá al usuario que lo seleccione y, en cuanto se seleccione, se procederá a iniciar la vinculación, que si resulta exitosa, enviará al microcontrolador el mensaje de conexión y permitirá el acceso a la pantalla del menú principal, la cual se visualizará junto al mensaje “Connected”. Si sucede algún error durante el proceso de vinculación, se mostrará un mensaje que contendrá la descripción del error que se ha producido.

En la Ilustración 6.2 se pueden observar los elementos descritos y su ordenación en la pantalla de vinculación.

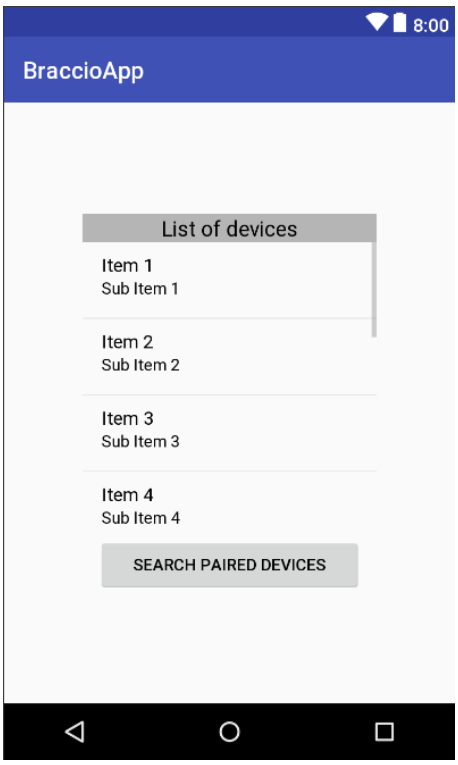


Ilustración 6.2. Pantalla de vinculación con el robot

6.1.2. Menú principal

La función de esta ventana es principalmente presentar las funcionalidades generales de la aplicación y permitir la navegación hacia ellas a partir de la selección del botón que contiene su nombre.

Por lo tanto, puesto que hay dos funciones básicas, la pantalla se compone de dos botones: uno para ingresar a la pantalla de control o "Jogging" y, el otro, para acceder a la pantalla de programación.

Como complemento a estos botones, se añadirá el texto "Tinkerkit Braccio Robot Control" y bajo él, una imagen del robot, de manera que la pantalla tenga algo de color y no quede tan vacía.

A continuación, se muestra la disposición de los elementos descritos en la pantalla.

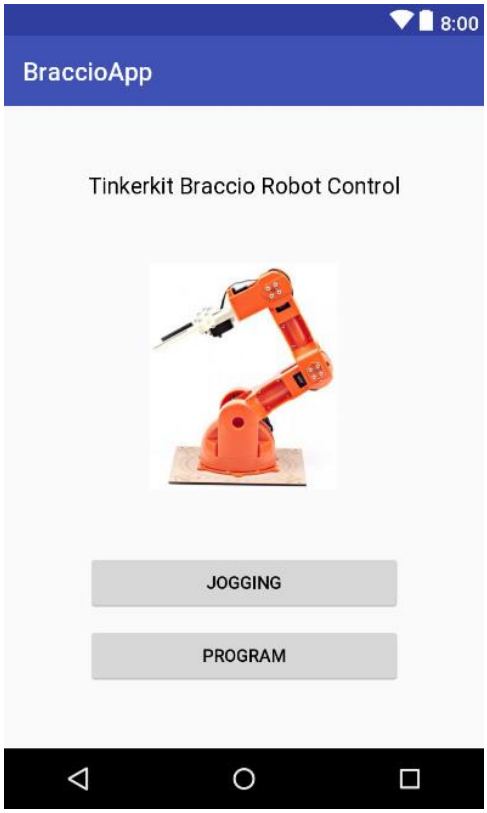


Ilustración 6.3. Pantalla del menú principal

6.1.3. Pantalla de control o "Jogging"

Se denomina "Jogging" al control manual del robot desde algún tipo de interfaz como las consolas de programación de la Ilustración 2.10 o el móvil, como es el caso. Esto se puede hacer de dos maneras a elegir por el usuario y que son a través del movimiento por articulación y el movimiento por coordenadas.

En el movimiento por articulación o “JOINT Jogging”, se selecciona la articulación que se desea mover y se le incrementa o decrementa el ángulo en el que se encuentra a través de los elementos de interfaz de usuario que incluya el dispositivo de control usado. Este tipo de movimiento es directo, es decir, no se requiere de cálculos previos para llevarlo a cabo.

En el control de movimiento por coordenadas XYZ, se selecciona el eje por el que se desea desplazar el elemento terminal, en este caso, la pinza, teniendo en cuenta que el sistema de coordenadas es cartesiano y de origen en la base del robot. En contraposición al tipo de control de movimiento anterior, éste presenta una complejidad mayor, ya que antes de poder mover el robot se han de realizar una serie de cálculos de manera que su resultado sea la posición que ha de tomar cada articulación del robot.

Después de llevar el robot a una posición de interés, es necesario poder guardar esta posición con una etiqueta que la identifique, de manera que, posteriormente, se pueda usar en la programación textual sin haber de especificar los ángulos de cada articulación para esa posición, cosa que responde al método de programación híbrida escogida.

Se pueden distinguir así tres funcionalidades dentro del “Jogging” (“Teaching” si se tiene en cuenta que se le enseñan al robot ciertas posiciones o configuraciones articulares), lo cual implementará a través de tres pestañas o comúnmente llamadas “tabs”, que se denominarán “JOINT” (“TAB1”), “XYZ” (“TAB2”) y “Positions” (“TAB3”).

En la pestaña “JOINT” se presentará una lista de seis elementos, en la que cada uno contiene un texto que especifica la articulación, un botón para restar grados a los ángulos de la articulación, otro botón para sumarlos y un texto que señala la posición articular actual y que se irá actualizando a medida que se mueva el robot. Cabe destacar que el robot responderá sólo al primer botón pulsado y que se moverá mientras esté ese botón presionado, lo que se traduce en que cuando se presione un botón, se enviará al microcontrolador una señal para aumentar o disminuir el ángulo de la articulación y en cuanto se libere, se enviará otra señal para parar.

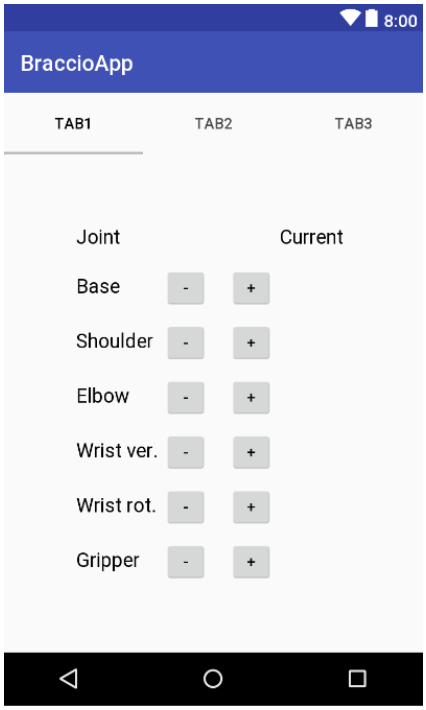


Ilustración 6.4. Pantalla "Jogging, mode JOINT mode"

La pestaña "XYZ" sigue el mismo posicionamiento y lógica que la pestaña "JOINT". La diferencia entre ellas radica en la lógica implementada en el microcontrolador, la cual ha de realizar los cálculos mencionados para poder mover el robot a través de un eje.

Ilustración 6.5. Pantalla "Jogging, XYZ mode"

Además de poder guardar las posiciones articulares actuales, se cree conveniente poder gestionarlas, lo que quiere decir, poder eliminarlas, ver las que se han guardado y ver el contenido de cada una de ellas. Esta gestión se implementará en una pestaña llamada "Position" que contiene un cuadro de texto en el que se podrá introducir el nombre de la posición que se desea gestionar, un botón que la guarda, un botón que muestra su información, un botón para eliminarla y dos textos que muestran la información contenida en la posición seleccionada y el nombre de las posiciones guardadas, respectivamente. La disposición de estos elementos se muestra en la siguiente ilustración.

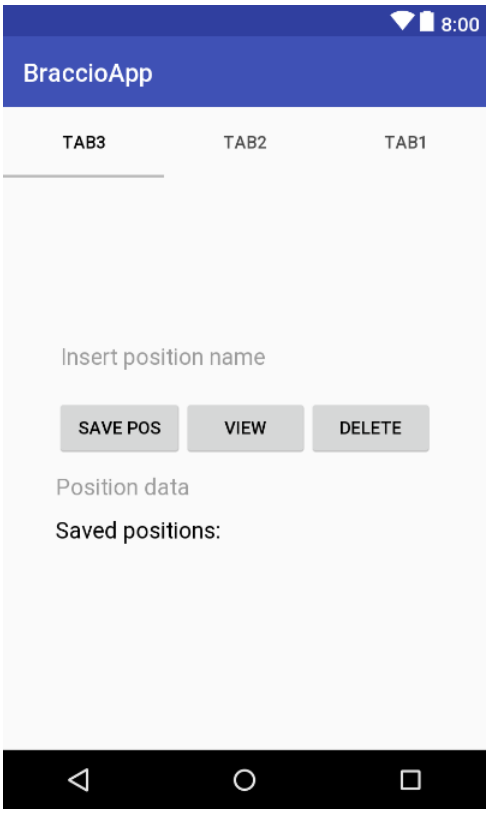


Ilustración 6.6. Pantalla "Jogging", modo de gestión de posiciones articulares

Otro detalle a comentar es que los nombres de las posiciones serán numéricas y corresponderán a la posición que ocupan dentro de una tabla de enteros. De esta manera, si se introduce el nombre de una posición que no existe, un mensaje avisará al usuario del rango de opciones que hay disponibles.

6.1.4. Pantalla de programación

En la pantalla de programación se podrá encontrar de un cuadro de texto en el que se podrán insertar los comandos de la tabla 3.1, los cuales han de ir en mayúscula, terminados en punto y coma (;) y un comando por línea.

Para poder saber si el texto introducido es correcto, se añadirá un botón de procesamiento que, al ser seleccionado, comprobará el código línea por línea y, si es correcto, lo codificará en los códigos

explicados en el apartado 5.2 “Protocolo de comunicaciones”. En caso de no ser correcto, se mostrarán los siguientes mensajes de error:

- “Insert a command at line _”: La línea indicada, aunque acaba en punto y coma, no contiene ningún otro dato, por lo que se pide al usuario que inserte un comando.
- “Missing ; at line _”: En la línea indicada, falta un punto y coma al final de la instrucción.
- “No such command at line _”: En la línea indicada, la primera palabra de la instrucción (comando) no existe dentro del set predefinido. Este error también se puede dar si se introduce más de una instrucción en la misma línea.
- “Too many arguments at line _”: En la línea indicada, aunque el comando es correcto, el número de argumentos introducidos es mayor del que se requiere.
- “Missing arguments at line _”: En la línea indicada, aunque el comando es correcto, el número de argumentos es menor de los que debería haber.
- “The second argument does not match with de command MOV at line _”: En la línea indicada, el segundo argumento del comando MOV no es un punto P_, un motor M_ o un eje válidos.
- “No such motor at line _”: En la línea indicada, que contendrá el comando MOV, el motor M_ indicado como segundo argumento no es ninguno de los disponibles (consultar tabla 5.4).
- “Missing angle at line _”: En la línea indicada, en el argumento que representa el ángulo en el comando MOV, donde debería haber un número, hay un carácter o conjunto de caracteres.
- “Invalid angle at line _”: En la línea indicada, el ángulo al que se desea mover un motor especificado y válido, ya que no se encuentra dentro del rango disponible.
- “Missing value at line _”: En la línea indicada, en el argumento en que debería haber un número, hay un carácter o conjunto de caracteres.
- “Insert a valid position at line _”: En la línea indicada, el nombre de la posición articular no es un número.
- “The selected position at line _ does not exist”: En la línea indicada, la posición articular a la que se quiere mover el brazo articulado no existe en memoria, por lo que no es válido.

Con la finalidad de enviar los códigos resultantes del texto procesado al microcontrolador para que éste lleve a cabo las órdenes deseadas, se incluirá el botón “Send”, que se activará una vez el texto introducido sea correcto y se desactivará mientras el programa esté en ejecución. Por defecto, al entrar a la pantalla de programación, este botón estará desactivado.

En el caso de que ocurriera alguna eventualidad durante la ejecución del programa, se dispondrá del botón “Stop”, el cual ha de enviar la orden de paro al microcontrolador y mostrar, a través de un pequeño mensaje, la línea en la que se ha detenido el programa. A su vez, ha de reiniciar el contador de programa para que se vuelva a comenzar desde el principio, si así lo deseara el usuario, y activar el botón “Send”.

Por último, con el objetivo de poder guardar los códigos realizados y validados y las posiciones guardadas en el momento de la creación del código, se incluirá la función de guardar el código en archivos de texto locales, los cuales también se podrán cargar y modificar. Para la realización de esta tarea será necesario un cuadro de texto, en el que se habría de introducir el nombre del archivo que se manipulará, y dos botones, uno con la finalidad de guardar el texto y el otro para cargar un archivo ya creado.

El proceso de guardar un archivo pasará por la verificación del texto que se desea guardar y la comprobación de que se ha introducido el nombre que se le dará al archivo que se guardará. En el caso de no verificar el código o no haberle dado un nombre al archivo, se mostrará un mensaje que avisará al usuario de estas circunstancias.

Respecto al proceso de importación o carga de un archivo ya creado, después de verificar si se ha especificado un nombre del archivo a manipular y si éste existe, se copiará el código del archivo en el cuadro de texto en el que se introduce el código, se borrarán todas las posiciones actuales guardadas y se cargarán las que contenga el archivo.

A continuación, se muestra una captura de pantalla en la que se observa la disposición de todos los elementos de los que consta la pantalla de programación explicados en esta sección.

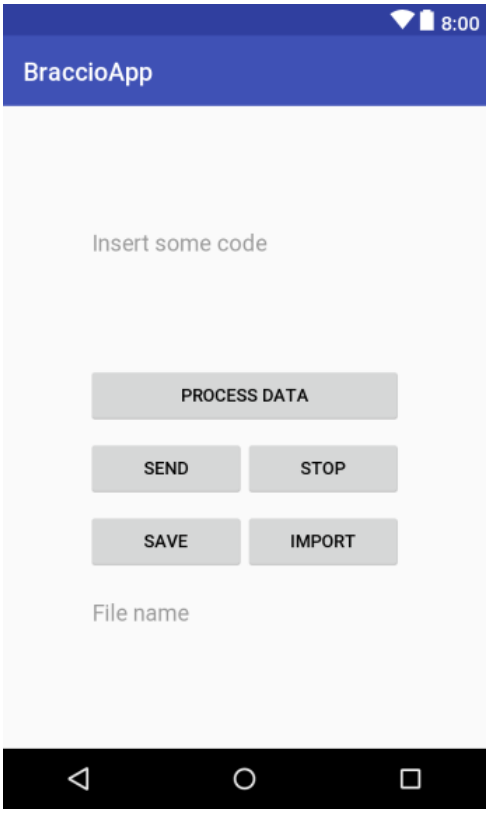


Ilustración 6.7. Pantalla de programación

6.1.5. Relación de pantallas

En la siguiente ilustración se pueden observar las pantallas de las que consta la aplicación de control del robot Braccio para móvil y las acciones que las enlazan, de manera que se entienda mejor la estructura y navegación a través de éstas.

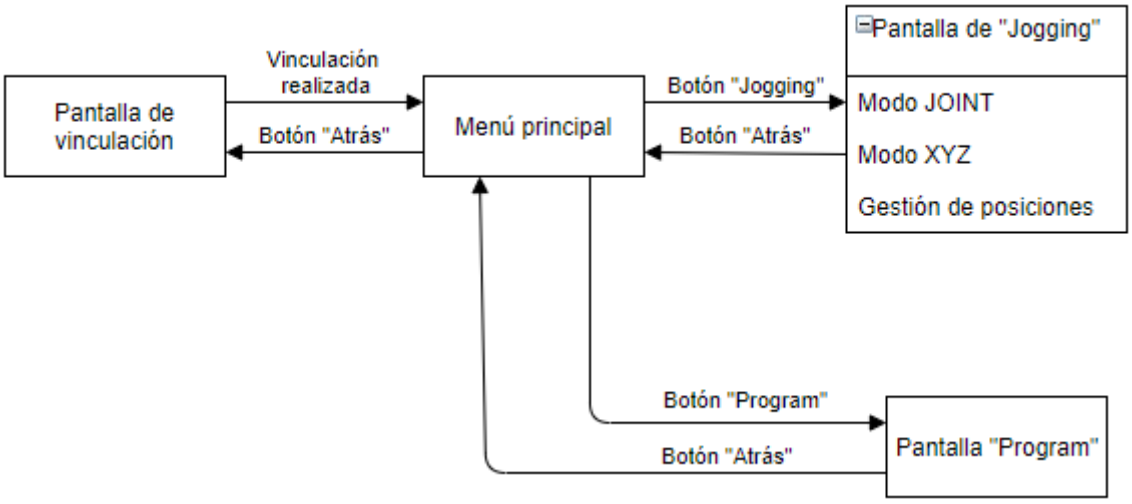


Ilustración 6.8. Diagrama de relación de pantallas

Cabe resaltar que el botón “Atrás”, que enlaza todas las pantallas, no está implementado como un botón dentro de la aplicación, sino que se hace uso del propio botón que incluye el móvil.

6.2. Otros elementos

Además del diseño e implementación de las pantallas de las que se compone la aplicación, existirán otros elementos necesarios para su correcto funcionamiento. Estos elementos son dos entidades, una que gestiona la comunicación mediante Bluetooth y otra que contiene todas las variables globales que se requieran.

6.3. Código en Android Studio

El código final, resultante del desarrollo del diseño planteado, se encuentra adjunto en el Anexo B: Código implementado en Android Studio, donde se presentan dos archivos por pantalla:

- El código en lenguaje Java que describe el comportamiento de las pantallas.
- La estructura o “layout” de cada pantalla, guardada en un archivo XML.

De la misma manera que con el código implementado para el microcontrolador, los comentarios se mostrarán inglés, con el objetivo de llegar a un público mayor.

7. Conclusiones

La realización de este trabajo ha supuesto un aumento considerable de los conocimientos en programación del autor, ya que se ha tenido que aprender el uso de dos lenguajes de programación muy diferentes entre ellos, cada uno con sus características propias, y de varios entornos de programación.

Durante el proceso de creación, se presentaron diversos inconvenientes, algunos de los cuales se presentarán a continuación y cuya resolución o gestión ha supuesto una oportunidad más para aprender más cosas.

En primer lugar, las limitaciones en el lenguaje C adaptado de Arduino, hace que su programación sea muy condicionada y algo confusa, si se tienen conocimientos previos de programación en C. Por esta razón, mientras se realiza el programa, es recomendable tener a mano la página web de Arduino.

Por otra parte, en un principio, se planteó usar el entorno de programación Qt y el lenguaje de programación JavaScript para crear la aplicación móvil, pero, después de un tiempo, resultó ser que para poder realizar proyectos de mayor complejidad, el lenguaje de programación debía ser cambiado a C++ adaptado a Qt, además de haber de comprar la versión completa. Por estas razones se decidió finalmente usar Android Studio, el cual es gratuito, cuenta con una gran comunidad de usuarios y resulta fácil adaptarse a su entorno.

Otro inconveniente a comentar es el hecho de no tener en cuenta las limitaciones de memoria, cosa que puede incurrir en resultados inesperados. En este caso, lo que sucedió es que, el procesamiento de las órdenes se realizaba en el microcontrolador y no en la interfaz, además del hecho de que los comandos no estaban codificados y conformaban cadenas de texto de larga extensión. Esta era una primera aproximación de lo que se pretendía realizar. El resultado de esto llegó al punto de sobrescribir memoria de programa. Las soluciones a este percance pasaban por cambiar de placa de desarrollo por otra con más memoria, añadir más memoria a la actual o replantear el algoritmo y comportamiento general actual.

Una de las especificaciones que se habían planteado es la posibilidad de guardar los programas del robot en memoria de la placa Arduino. A causa de lo anterior, esta funcionalidad dejó de ser una opción. Se tuvo en cuenta el uso de un módulo para tarjeta SD, pero el pinaje de la placa actual no lo permitió, además de ser una opción poco recomendable, al haberse de cargar todo le texto de un archivo seleccionado en la memoria del microcontrolador. La mejor opción es la propuesta que se ha sugerido como ampliación de este trabajo y que consiste en el diseño e implementación de una interfaz electrónica que permita las mismas acciones que el móvil.

Como se puede observar, a pesar de todos los inconvenientes surgidos, el objetivo del trabajo ha sido conseguido y su resultado es satisfactorio, tanto por cumplir con las especificaciones planteadas como por todo lo que ha aportado su desarrollo.

8. Estudio económico

Los puntos clave de este trabajo son que está realizado de manera que no se hayan de poseer conocimientos técnicos previos y que supone una inversión mucho menor respecto a la adquisición de un robot manipulador industrial con el cual aprender programación de robots. Esto contribuye a que más personas se puedan acercar al mundo de la robótica industrial. De hecho, con un poco más de desarrollo podría usarse en las aulas de las escuelas, para cualquier franja de edad.

A continuación se presentará el coste desglosado que ha supuesto el desarrollo de este trabajo.

8.1. Coste de los recursos usados

Tabla 8.1. Coste de los recursos usados

Producto	Cantidad	Precio (€)	Total (€)
Ordenador Portátil	1	350	350
Microsoft Visual Studio	1	0	0
Arduino IDE	1	0	0
Android Studio	1	0	0
Teléfono Android	1	80	80
Tinkerkit Braccio Robot	1	199	199
Módulo de Bluetooth HC-06	1	9,6	9,6
Arduino UNO Rev3	1	20	20
TOTAL			658,6

8.2. Coste por mano de obra

Tabla 8.2. Coste por mano de obra

Tarea	Horas	Precio(€)/Hora	Total (€)
Diseño de la aplicación de Arduino	20	20	400
Diseño de la aplicación de Android	15	20	300
Programación de la aplicación de Arduino	70	20	1400
Programación de la aplicación de Android	40	15	600
Montaje del robot Braccio	5	10	50
TOTAL			2750

En el desglose presentado, se puede ver que el trabajo de diseño, propio de la ingeniería, suma 35 horas.

La programación de la placa de desarrollo Arduino UNO, se realiza con un lenguaje de bajo nivel (C++), el cual requiere de más conocimientos informáticos y de más dedicación, por lo cual, las horas invertidas en esta tarea son más caras que las de la programación de la aplicación para móvil Android, cuyo lenguaje es Java.

Por último, el montaje del robot Braccio es una tarea propia de un técnico, cuyas horas son más baratas que la de un ingeniero.

8.3. Resumen de costes

Sumando todos los costes descritos anteriormente, se obtiene que el precio total de la realización de este trabajo es de 3408,6 €.

Este es un precio que resulta algo alto, pero en el caso de producir una versión para la venta a partir de este trabajo, los costes de diseño y programación se repartirían entre las ventas esperadas.

Teniendo esto en cuenta y suponiendo unas ventas de 250 kits, el precio de poder obtener un robot programable desde el móvil de una forma parecida a la de los robots industriales sería aproximadamente de 375 €, lo que sigue siendo muy inferior al coste de la adquisición y mantenimiento de un robot manipulador industrial.

9. Bibliografía

[1] Arduino Store, Tinkerkit Braccio Robot. [En línea]: https://store.arduino.cc/tinkerkit-braccio [Consulta: abril 2018].

[2] ARI_T1_Introducción.pdf (2015). Escuela de Ingeniería de Barcelona Este, Barcelona: Sebastián Tornil, pp.3, 7, 16, 17.

[3] Es.wikipedia.org (2018). R.U.R. (Robots Universales Rossum). [En línea] Disponible en: https://es.wikipedia.org/wiki/R.U.R._(Robots_Universales_Rossum) [Consulta: abril 2018].

[4] Sisbib.unmsm.edu.pe (2001). Actualidad y perspectiva de la robótica. [En línea] Disponible en: http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/indata/v04_n1/actualidad.htm#Introducción [Consulta: abril 2018].

[5] Industrial Robots 2016 Chapter 1_2.pdf. (2016). [eBook] pp.1, 2, 3, 5. Disponible en: https://ifr.org/img/office/Industrial_Robots_2016_Chapter_1_2.pdf [Consulta: abril 2018].

[6] Ifr.org (2016). International Federation of Robotics. [En línea] Disponible en: http://www.ifr.org/service-robots/ [Consulta: abril 2018].

[8] RobotWorx (2018). "What Are Gantry Robots?" [En línea] Disponible en: https://www.robots.com/faq/what-are-gantry-robots [Consulta: abril 2018].

[9] iRobot Roomba 980 [imagen] Disponible en: http://www.irobotweb.com/~/media/EU-Sites/Store/Products/whyRoomba.jpg?h=215&la=es-ES&w=339 [Consulta: abril 2018].

[10] Ecovacsrobotics, Winbot W710 (2015) [imagen] Disponible en: https://ecovacsrobotics.com/wordpress/wp-content/uploads/2015/06/ecovacs-robotics-winbot-w710-robot_underside.jpg [Consulta: abril 2018].

[11] Dolphinpoolrobot, Robot limpiafondos Dolphin Supreme M5] [imagen] Disponible en: http://dolphinpoolrobot.com/images/phocagallery/supreme_m5/thumbs/phoca_thumb_l_bottom.jpg [Consulta: abril 2018].

[12] El Mundo. Robot Zowi de BQ (2016) [imagen] Disponible en: http://e04-elmundo.uecdn.es/assets/multimedia/imagenes/2015/10/14/14448369146415.jpg [Consulta: abril 2018].

[13] KUKA Coaster (2018) [imagen] Disponible en: http://www.kuka-robotics.com/res/sps/9cb8e311-bfd7-44b4-b0ea-b25ca883f6d3_KUKA_Coaster_EN.pdf [Consulta: abril 2018].

[14] Cyberdyne, HAL para uso médico (2018) [imagen] Disponible en: http://www.cyberdyne.jp/english/products/images/lowerlimb_medical/img_specifications_01.jpg [Consulta: abril 2018].

[15] Service Robots 2016 Chapter 1_2.pdf. (2016) [eBook] pp.1, 2, 3, 4. Disponible en: https://ifr.org/img/office/Service_Robots_2016_Chapter_1_2.pdf [Consulta: abril 2018].

[16] DaVinci Surgery (2016) [imagen] Disponible en: http://davincivscancer.com/files/pagina/original/20140807120854.png [Consulta: abril 2016].

[17] Consolas de programación (2018) [imagen] Disponible en: https://www.foxon.cz//img/cms/robot_kuka_abb_fanuc_opravy_foxon.jpg [Consulta: abril 2018].

[18] ARI_T5_Programación.pdf. (2015). Escuela de Ingeniería de Barcelona Este, Barcelona: Sebastián Tornil, pp.6, 7, 8, 9, 10, 11, 13, 14.

[19] Colman, R. (2016) Robotics meets the mainstream [En línea] Canadianmetalworking.com Disponible en: https://www.canadianmetalworking.com/article/automationsoftware/robotics-meets-mainstream [Consulta: abril 2018].

[20] Features RT ToolBox3 Software Industrial Robots-MELFA | MITSUBISHI ELECTRIC FA. [En línea] Disponible en: http://www.mitsubishielectric.com/fa/products/rbt/robot/smerit/rt3/index.html [Consulta: abril 2018].

[21] Arduino Project Hub (2016) Control your RoboArm with your Smartphone [En línea] Disponible en: https://create.arduino.cc/projecthub/zone-team/control-your-roboarm-with-your-smartphone-b61da6?ref=search&ref_id=mearm&offset=5 [Consulta: abril 2018].

[22] Arduino Project Hub (2016) MeArm Robot Arm - Your Robot - V1.0 [En línea] Disponible en: https://create.arduino.cc/projecthub/benbobgray/mearm-robot-arm-your-robot-v1-0-326702?ref=search&ref_id=mearm&offset=0 [Consulta: abril 2018].

[23] Arduino Project Hub (2017) Robotic Arm from Recycled Materials [En línea] Disponible en: https://create.arduino.cc/projecthub/circuito-io-team/robotic-arm-from-recycled-materials-7e318a?ref=search&ref_id=recycled%20materiasl&offset=0 [Consulta: abril 2018].

[24] Arduino Project Hub (2017) Arduino IoT Robotic Arm [En línea] Disponible en: https://create.arduino.cc/projecthub/aerdronix/arduino-iot-robotic-arm-5a4401?ref=search&ref_id=iot%20robotic%20arm&offset=1 [Consulta: abril 2018].

[25] Arduino Project Hub (2016) Control Arduino Robot Arm with Android App [En línea] Disponible en: https://create.arduino.cc/projecthub/slantconcepts/control-arduino-robot-arm-with-android-app-1c0d96?ref=search&ref_id=arm%20android%20app&offset=3 [Consulta: abril 2018].

[26] Robotic Arm Edge (2018) Robotic Arm Edge [En línea] OWI Inc. dba: Robotikits™ Direct. Disponible en: http://www.owirobot.com/robotic-arm-edge-1/ [Consulta: abril 2018].

[27] Braccio Quick Start Guide (2016) 2nd ed. [eBook] Scarmagno, Italia: ARDUINO S.r.l. Disponible en: https://docs-emea.rs-online.com/webdocs/14da/0900766b814da22f.pdf [Consulta: abril 2018].

[28] Contenido del Tinkerkit Braccio Robot [imagen] Disponible en: https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/500x375/f8876a31b63532bbba4e781c30024a0a/t/0/t050000_unboxed_1.jpg [Consulta: abril 2018].

[29] Configuración del robot Braccio (2018) [imagen] Disponible en: https://media.rs-online.com/t_large/F1113738-03.jpg [Consulta: abril 2018].

[30] ATmega328/P Complete Datasheet (2016) [eBook] San Jose, California: Atmel Corporation. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf [Consulta: abril 2018].

[31] Arduino Store (2018) Arduino Uno Rev3. [En línea] Disponible en: https://store.arduino.cc/arduino-uno-rev3 [Consulta: abril 2018].

[32] Aprendiendo Arduino (2016) Arduino Uno a fondo. Mapa de pines. [En línea] Disponible en: https://aprendiendoarduino.wordpress.com/2016/06/27/arduino-uno-a-fondo-mapa-de-pines-2/ [Consulta: abril 2018].

[33] Prometec.net (2017) Módulo BlueTooth HC-06 | Tienda y Tutoriales Arduino. [En línea] Disponible en: https://www.prometec.net/bt-hc06/ [Consulta: abril 2018].

[34] Martyn Currey (2014) [En línea] Disponible en: http://www.martyncurrey.com/wp-content/uploads/2014/10/HC-06_zs-040_01_1200.jpg [Consulta: abril 2018].

[35] e-Gizmo Mechatronix Center, Comandos AT. [En línea] Disponible en: https://docs.google.com/file/d/0B93S-Z6qmW9ER3hyRmNIazd4bVE/edit [Consulta: abril 2018].

[36] Cytron Technologies (2018) Bluetooth Serial Transceiver HC06 [En línea] Disponible en: https://www.cytron.io/p-bt-hc-06 [Consulta: abril 2018].

[37] Arduino Shield para Tinkerkit Braccio Robot (2018) [imagen] Disponible en: https://www.rs-online.com/designspark/rel-assets/ds-assets/uploads/images/5791fbff4e1c48d89daa2d300ab5c436Shield.JPG [Consulta: abril 2018].

[38] Google Play (2013) "BlueTerm App". [En línea] Disponible en: https://play.google.com/store/apps/details?id=es.pymasde.blueterm&hl=es_419 [Consulta: abril 2018].

[39] Android Developers (2018) Android Developers [En línea] Disponible en: https://developer.android.com/index.html [Consulta: abril 2018].

[40] Arduino (2018) Arduino - Home. [En línea] Disponible en: https://www.arduino.cc/ [Consulta: abril 2018].

[41] Arduino Forum (2018) Arduino Forum - Index [En línea] Disponible en: https://forum.arduino.cc/ [Consulta: abril 2018].

[42] Fritzing (2018) Fritzing [En línea] Disponible en: http://fritzing.org/learning/ [Consulta: abril 2018].

Anexo A: Código implementado en Arduino

A1. Inicio.ino

```
#include "BraccioControl.h"

// Initialization of the braccioControl object
BraccioControl braccio;

void setup()
{
	// Initialization of Braccio object
	braccio.Start();
	Serial.println("End of initialization");
}

void loop()
{
	// Data update of braccio object
	braccio.Update();

}
```

A2. BraccioControl.h

```
#ifndef __BRACCIOCONTROL__
#define __BRACCIOCONTROL__

#include "Modules.h"
#include "RobotModule.h"
#include "ProcessDataModule.h"
#include "ComsModule.h"

#define MODULES 3

class BraccioControl
{
public:
	// Constructor
	BraccioControl();

	// Modules initialization
	void Start();

	// Modules update
	void Update();

	// Explicit set of the minimum delay between movements of 1 degree (10 - 30
ms)
	void SetDelay(int step_delay);

public:
	// Variabless declaration
	ComsModule coms_module;
	ProcessDataModule process_data;
	RobotModule robot;

	Module* modules[MODULES];
};

extern BraccioControl braccio;

#endif
```

A3. Braccio.cpp

```
#include "BraccioControl.h"

BraccioControl::BraccioControl()
{
	modules[0] = &coms_module;
	modules[1] = &process_data;
	modules[2] = &robot;
}

void BraccioControl::Start()
{
	for (int i = 0; i < MODULES; ++i)
		modules[i]->Start();
}

void BraccioControl::Update()
{
	for (int i = 0; i < MODULES; ++i)
		modules[i]->Update();
}

void BraccioControl::SetDelay(int step_delay)
{
	robot.SetDelay(step_delay);

}
```

A4. Modules.h

```
#ifndef __MODULE__
#define __MODULE__

class Module
{
public:
	// Each module has to have a Start and Update method.

	// Constructor
	Module() {}

	// Module initialization
	virtual void Start() {}

	// Module update
	virtual void Update() {}
};

#endif
```

A5. ComsModule.h

```
#ifndef __COMSMODULE__
#define __COMSMODULE__

#include "Modules.h"
#include "Communication.h"
#include "BluetoothModule.h"
#include "SerialModule.h"

class ComsModule : public Module
{
public:
	// Enumeration that contains the index of the available communication types
	enum CommTypes {
		CT_SERIAL,
		CT_BLUETOOTH,
		CT_NULL
	};
public:
	// Constructor
	ComsModule();

	// Communications initialitzation (Serial, bluetooth)
	void Start();

	// Communications update
	void Update();

	// Function that enables to send any text to any possible communication in
connected state
	void SendData(String data, CommTypes _communication);

	// Function that sets the disconnect state to the communication passed as
argument
	void DisconnectComs(CommTypes _communication);

	// Function that sets the connect state to the communication passed as
argument
	void ConnectComs(CommTypes _communication);

	// Function that returns the current state to the communication passed as
argument
	int GetComsState(CommTypes _communication);

private:

	// Variables declaration
	Communication * coms[CT_NULL];
	BluetoothModule bt_module;
	SerialModule serial_module;
};

#endif
```

A6. ComsModule.cpp

```
#include "ComsModule.h"
#include "BraccioControl.h"
#include "ProcessDataModule.h"

ComsModule::ComsModule()
{
	coms[0] = &serial_module;
	coms[1] = &bt_module;
}

void ComsModule::Start()
{
	// For each available communication type, a Start() method will be called
	for (int i = 0; i < CT_NULL; i++) {
		coms[i]->Start();
	}
}

void ComsModule::Update()
{
	// According to the current state of each communication type, if they are
disabled, an
	// attempToConnect method will verify if it's activated again. Otherwise, if
the current
	// state is connected, the reception of data from them will be monirtorized.
	// When some data is received, it goes to the ProcessDataModule in order to
be processed.
	for (int i = 0; i < CT_NULL; i++) {
		String received_data = "";

		switch (coms[i]->GetState())
		{

		case Communication::S_CONNECTED:
		{
			received_data = coms[i]->GetData();
			if (received_data != "") {
				if (serial_module.GetState() ==
Communication::S_CONNECTED) {

					// Uncomment in case of need some debbuging at
this part

					/*Serial.print(i);
					Serial.print(" Received --> ");
					Serial.println(received_data);*/
				}
				braccio.process_data.ProcessData(received_data);
			}
			break;
		}
		case Communication::S_DISCONNECTED:
			coms[i]->AttemptToConnect();
			break;
		case Communication::S_NULL:
			break;
```

```
		default:
			break;
		}
	}
}

void ComsModule::SendData(String data, CommTypes _communication)
{
	// If the communication is connected, then the message can be sent

	if (_communication < CT_NULL && _communication >= 0) {
		if (GetComsState(_communication) == Communication::S_CONNECTED)
			coms[_communication]->SendData(data);
	}
}

void ComsModule::DisconnectComs(CommTypes _communication)
{
	if (_communication < CT_NULL && _communication >= 0) {
		coms[_communication]->SetState(Communication::S_DISCONNECTED);
		if (_communication == 1) bt_module.SetBluetoothStateDisconnected();
	}
}

void ComsModule::ConnectComs(CommTypes _communication)
{
	if (_communication < CT_NULL && _communication >= 0) {
		coms[_communication]->SetState(Communication::S_CONNECTED);
	}
}

int ComsModule::GetComsState(CommTypes _communication)
{
	Communication::State state = Communication::S_NULL;

	if (_communication < CT_NULL && _communication >= 0) {
		state = coms[_communication]->GetState();
	}

	return state;

}
```

A7. Communication.h

```
#ifndef __COMMUNICATION__
#define __COMMUNICATION__

#include "Arduino.h"

class Communication
{

	// Each communication type has a curr_state variable and a Start, SendData,
	// AttempToConnect and GetData methods. Also, the curr_state can be set and
get.

public:
	enum State {
		S_CONNECTED,
		S_DISCONNECTED,
		S_NULL
	};

public:
	// Constructor
	Communication() {}

	// Communication type initialization
	virtual void Start() {}

	// Method that allows to send specified data through the communication type
	virtual void SendData(String data) {}

	// Method that checks if the communication type becomes actived.
	virtual void AttemptToConnect() {}

	// Method that checks the reception of data
	virtual String GetData() { return ""; }

	// Function that returns the current state to the communication type
	State GetState() { return curr_state; }

	// Function that sets the current state to the communication type
	void SetState(State state) { curr_state = state; }

private:
	// Variables declaration
	State curr_state = S_NULL;
};


#endif
```

A8. BluetoothModule.h

```
#ifndef __BLUETOOTHMODULE__
#define __BLUETOOTHMODULE__

#include "Communication.h"
#include <SoftwareSerial.h>
#include "Arduino.h"

class BluetoothModule : public Communication
{

public:
	// Constructor
	BluetoothModule();

	// Initialitzacion of the Bluetooth module
	void Start();

	// Implementation of the communication type method SendData, that enables
	// to send especified data through the Bluetooth module
	void SendData(String data);

	// Method that sets the Bluetooth current state to disconnected
	void SetBluetoothStateDisconnected();

	// Manages the current state of the connection
	void AttemptToConnect();

	// Enables to get a string of data instead of a single character from the
module
	String GetData();


private:

	// Variables declaration
	unsigned long time_stamp = 0;
	bool disconnect_checked = false;
	State curr_state = S_NULL;

};

#endif
```

A9. BluetoothModule.cpp

```
#include "BluetoothModule.h"

// RX y TX pins will be set at digitals pins 4 and 2
SoftwareSerial BT(4, 2);

BluetoothModule::BluetoothModule()
{
}

void BluetoothModule::Start()
{
	BT.begin(9600);
	SetState(S_DISCONNECTED);

	// At the initialization, AT command is sent in order to check if the
Bluetooth module exists
	BT.print("AT");
	time_stamp = millis();
}

void BluetoothModule::SendData(String data)
{
	// The data to be sent will be displayed in the Arduino console
	Serial.println("To BT -->" + data);

	// Uncomment if you need to send AT commands, and comment the other one
	//BT.print(data);

	// Sending of specified data ended with de # char
	BT.print(data + "#");
}

void BluetoothModule::SetBluetoothStateDisconnected()
{
	SetState(S_DISCONNECTED);
	disconnect_checked = false;
}

void BluetoothModule::AttemptToConnect()
{
	// If the reception of data is detected, the data will be managed
	// in order to know the state of the conection
	if (BT.available())
	{
		String received_data = GetData();

		// Checks once if the Bluetooth module exists
		// Checks if the Bluetooth module is actived
		if (received_data == "OK" && !disconnect_checked)
		{
			SetState(S_DISCONNECTED);
			disconnect_checked = true;
			Serial.println(received_data);
			Serial.println("Bluetooth no connected.");
		}
		else
```

```
                {
                        Serial.println("Bluetooth connected.");
                        SetState(S_CONNECTED);
                }
        }

        // If no data has been received, this part checks every second the presence
        // of the Bluetooth module
        else if ((millis() - time_stamp > 1000) && !disconnect_checked)
        {
                BT.print("AT");
                time_stamp = millis();
        }
}

String BluetoothModule::GetData()
{
        String tmp_data = "";

        // If the reception of data is detected, each character will be read and
        // concatenated until the end of line ( char(-1) or # )
        if (BT.available())
        {
                char new_char = BT.read();
                while (new_char != -1)
                {
                        if (new_char == '#') break;
                        tmp_data += new_char;
                        delay(25);
                        new_char = BT.read();
                }
        }

        return (tmp_data);
}
```

A10. SerialModule.h

```
#ifndef __SERIALMODULE__
#define __SERIALMODULE__

#include "Communication.h"
#include "Arduino.h"

class SerialModule : public Communication
{
public:
	// Constructor
	SerialModule() {}

	// Serial initialitzation
	void Start();

	// Implementation of the communication type method SendData, that enables
	// to send especified data to the Serial console
	void SendData(String data);

	// Function that runs while ComsModule Update methode is running
	// and checks if the Serial is enabled. If so, sets the connected state
	void AttemptToConnect();

	// Enables to get a string of data instead of a single character from the
Arduino console
	String GetData();
};

#endif
```

A11. SerialModule.cpp

```
#include "SerialModule.h"

void SerialModule::Start()
{
	Serial.begin(9600);
	SetState(S_DISCONNECTED);
}

void SerialModule::SendData(String data)
{
	Serial.println(data);
}

void SerialModule::AttemptToConnect()
{
	if (S_DISCONNECTED && Serial) {
		SetState(S_CONNECTED);
		Serial.println("Serial connected");
	}
}

String SerialModule::GetData()
{

	// If the reception of data is detected, each character will be read and
	// concatenated until the end of line char(-1)
	String tmp_data = "";
	if (Serial.available())
	{
		char new_char = Serial.read();
		while (new_char != -1)
		{
			tmp_data += new_char;
			delay(25);
			new_char = Serial.read();
		}
	}

	// Also, the acive connection is checked in order to disable sending data
through Serial
	// if it's disconnected
	if (S_CONNECTED && !Serial) {
		SetState(S_DISCONNECTED);
	}

	return (tmp_data);

}
```

A12. ProcessDataModule.h

```
#ifndef __PROCESSDATAMODULE__
#define __PROCESSDATAMODULE__

#include "Modules.h"
#include "CommandClass.h"
#include "List.h"
#include "Arduino.h"
#define N_COMMANDS 9

class ProcessDataModule : public Module
{
public:
	// Enumeration that contains the index of the available command types
	enum CommandTypes {
		C_CONNECT,
		C_DISCONNECT,
		C_REQUEST,
		C_SENDTO,
		C_JOGGING,
		C_MOVE,
		C_HAND,
		C_SAVEPOS,
		C_STOP,
		C_NULL
	};
public:
	// Constructor
	ProcessDataModule();

	// Start and Update methods are declared, for future usages
	void Start() {}
	void Update() {}

	// Any communicating module can send received data to be processed
	// in this method
	void ProcessData(String text);

private:
	// Separates each argument separated by a "separator" and put them in a
dinamic list
	void Tokenize(String tmp_data, p2List<int> &list, char separator);

private:
	// Variables declaration
	CommandClass * commands[N_COMMANDS];
	CommandClass* connect;
	CommandClass* disconnect;
	CommandClass* request_data;
	CommandClass* send;
	CommandClass* jogging;
	CommandClass* move;
	CommandClass* hand;
	CommandClass* save_pos;
	CommandClass* program_stop;
};
#endif
```

A13. ProcessDataModule.cpp

```
#include "ProcessDataModule.h"
#include "BraccioControl.h"
#include "BluetoothModule.h"
#include "RobotModule.h"
#include "ComsModule.h"

ProcessDataModule::ProcessDataModule()
{
	// Initialization of all available commands
	connect = new CommandClass(1, 1,
CommandClass::AvailableModules::M_CONNECTIVITY);
	commands[0] = connect;
	disconnect = new CommandClass(1, 1,
CommandClass::AvailableModules::M_CONNECTIVITY);
	commands[1] = disconnect;
	request_data = new CommandClass(1, 1,
CommandClass::AvailableModules::M_CONNECTIVITY);
	commands[2] = request_data;
	send = new CommandClass(1, 1,
CommandClass::AvailableModules::M_CONNECTIVITY);
	commands[3] = send;
	jogging = new CommandClass(2, 2, CommandClass::AvailableModules::M_ROBOT);
	commands[4] = jogging;
	move = new CommandClass(1, 2, CommandClass::AvailableModules::M_ROBOT);
	commands[5] = move;
	hand = new CommandClass(1, 1, CommandClass::AvailableModules::M_ROBOT);
	commands[6] = hand;
	save_pos = new CommandClass(1, 8, CommandClass::AvailableModules::M_ROBOT);
	commands[7] = save_pos;
	program_stop = new CommandClass(0, 0,
CommandClass::AvailableModules::M_ROBOT);
	commands[8] = program_stop;
}

void ProcessDataModule::ProcessData(String received_data)
{
	// If the first character is not number, it will be shown in the Arduino
console
	if (received_data.charAt(0) < 48 && received_data.charAt(0) > 57)
Serial.println(received_data);

	// If it is, the received data will be processed
	else {
		// Gets de command index
		int command = received_data.substring(0, 1).toInt();

		// Uncomment for debugging purpose

		/*Serial.print("command: ");
		Serial.println(command);*/

		// If the command index is valid, the received data will be tokenized
(separated and put into an integer
		// array) and the arguments will be checked since we have maximum and
minimum number of arguments that
		// has to have
```

```
            if (command >= 0 && command < C_NULL) {
                  p2List<int> tokens;
                  if (command == 3) Tokenize(received_data.substring(0, 3),
tokens, ' ');
                  else Tokenize(received_data, tokens, ' ');

                  // Uncomment for debugging purpose

                  /*for (int i = 0; i < tokens.count(); i++) {
                        Serial.println(tokens[i]);
                  }*/

                  bool commandfound = false;
                  CommandClass::AvailableModules type =
CommandClass::AvailableModules::M_NULL;
                  int num_args = tokens.count() - 1;

                  // Uncomment for debugging purpose

                  /*Serial.print("num_args: ");
                  Serial.println(num_args);*/

                  if (num_args >= commands[command]->GetMinArgs() && num_args <=
commands[command]->GetMaxArgs())
                  {
                        commandfound = true;
                        type = commands[command]->GetModuleType();
                  }

                  // If the command syntax is correct, the only thing to do is
sending the order to the
                  // module which handles it
                  if (commandfound)
                  {
                        switch (type)
                        {
                        case CommandClass::AvailableModules::M_NULL:
                              break;
                        case CommandClass::AvailableModules::M_ROBOT:
                              if (command == C_JOGGING)
braccio.robot.JoggingCommand(tokens);
                              else if (command == C_MOVE)
braccio.robot.MoveCommand(tokens);
                              else if (command == C_HAND)
braccio.robot.SetGripper(tokens[1]);
                              else if (command == C_SAVEPOS)
braccio.robot.SavePosition(tokens);
                              else if (command == C_STOP)
braccio.robot.StopProgram();
                              break;
                        case CommandClass::AvailableModules::M_CONNECTIVITY:
                              if (command == C_CONNECT) {

      braccio.coms_module.ConnectComs((ComsModule::CommTypes)tokens[1]);
                              }
                              else if (command == C_DISCONNECT) {

      braccio.coms_module.DisconnectComs((ComsModule::CommTypes)tokens[1]);
                              }
```

```
                                    else if (command == C_REQUEST) {

        braccio.coms_module.SendData(braccio.robot.BuildStringCurrentAngles(),
(ComsModule::CommTypes)tokens[1]);
                                    }
                                    else if (command == C_SENDTO) {

        braccio.coms_module.SendData(received_data.substring(4),
(ComsModule::CommTypes)tokens[1]);
                                    }
                                }
                            }
                        }
                }
        }

void ProcessDataModule::Tokenize(String tmp_data, p2List<int> &list, char separator)
{
        // Each character is concatenated and checked until it's equal to the
separator character, end of line or a line break
        // This Tokenize method is exclusive for the creation of integers lists
        tmp_data += char(-1);
        for (int i = 0; i < tmp_data.length(); i++)
        {
                String tmp = "";
                char tmp_char = tmp_data.charAt(i);
                while (tmp_char != -1 && tmp_char != separator && tmp_char != '\n')
                {
                        tmp += tmp_char;
                        i++;
                        tmp_char = tmp_data.charAt(i);
                }
                char vector_char[5];
                tmp.toCharArray(vector_char, 5);
                list.add(atoi(vector_char));
        }

}
```

A14. CommandClass.h

```
#ifndef __COMMANDCLASS__
#define __COMMANDCLASS__

class CommandClass
{
	// Each command class has a managing module related to the command behaviour,
	// a number of minimum arguments, a number of maximum arguments and the
methods
	// to get them

public:
	enum AvailableModules {
		M_CONNECTIVITY,
		M_ROBOT,
		M_NULL
	};

public:
	// Constructor
	CommandClass(int min_args, int max_args, AvailableModules type);

	// Method that returns the minimun arguments related to the command behaviour
	int GetMinArgs() const;

	// Method that returns the maximum arguments related to the command behaviour
	int GetMaxArgs() const;

	// Method that returns the managing module related to the command behaviour
	AvailableModules GetModuleType() const;

private:
	// Variables declaration
	int min_args;
	int max_args;
	AvailableModules type;
};

#endif
```

A15. CommandClass.cpp

```
#include "CommandClass.h"

CommandClass::CommandClass(int min_args, int max_args,
CommandClass::AvailableModules type)
{
	// Assignment of the passed variables to the command ones
	this->min_args = min_args;
	this->max_args = max_args;
	this->type = type;
}

int CommandClass::GetMinArgs() const
{
	return min_args;
}

int CommandClass::GetMaxArgs() const
{
	return max_args;
}

CommandClass::AvailableModules CommandClass::GetModuleType() const
{
	return type;

}
```

A16. RobotModule.h

```
#ifndef __ROBOTMODULE__
#define __ROBOTMODULE__

#define NUMBERPOSITIONS 20
#define ELEMENTSPOS 6

#include "Modules.h"
#include "RoboticArm.h"
#include "List.h"
#include "Arduino.h"

class RobotModule : public Module
{
public:
	// Constructor
	RobotModule();

	// Initialitzacion of Braccio
	void Start();

	// Update of Braccio variables
	void Update();

	// Explicit set of the delay between single movements of 1 degree
	void SetDelay(int step_delay);

	// Method that sets the maximum or minimum value to the  gripper target
	// angle,in order to close or open the gripper, respectively
	void SetGripper(int gripper_state);

	// Method that manages the manual control of Braccio movements
	void JoggingCommand(p2List<int> &list);

	// Method that manages the automatic control of Braccio movements
	void MoveCommand(p2List<int> &list);

	// Method that returns a string with the command "Response" followed by the
current
	// angles of all joints, all separated by an space character
	String BuildStringCurrentAngles();

	// Method that manages what to do with specified joint positions
	void SavePosition(p2List<int> &list);

	// Method that stops the execution of the current automatic movement
(program)
	void StopProgram();

private:
	// Variables declaration
	RoboticArm arm;
	int positions[NUMBERPOSITIONS*ELEMENTSPOS];
};

#endif
```

A17. RobotModule.cpp

```
#include "RobotModule.h"
#include "BraccioControl.h"
#include "ComsModule.h"

RobotModule::RobotModule()
{
}

void RobotModule::Start()
{
	//	Initialization of the robotic arm and set to -1 the values of all
available joint positions
	arm.Start();
	for (int i = 0; i < NUMBERPOSITIONS*ELEMENTSPOS; i++) {
		positions[i] = -1;

		// Uncomment for debbuging purpose

		/*Serial.print("Position ");
		Serial.print(i);
		Serial.print(": ");
		Serial.println(positions[i]);*/
	}
}

void RobotModule::Update()
{
	// Update of the robotic arm data, which means moving the robot until the
target angles are reached
	arm.Move();
}

void RobotModule::SetDelay(int step_delay)
{
	arm.SetDelay(step_delay);
}

void RobotModule::SetGripper(int gripper_state)
{
	// Set of the gripper target angles if possible.
	// If the possible states (0/1) are correct, if the current angles are equal
to target ones,
	// a flag of no movement will be sent to the movile app, if the connection
allows it
	// If the current angles are not equal to target ones, the new target angles
will be set
	if (gripper_state == 0) {
		if (arm.GetCurrentAngles(RoboticArm::JointTypes::JT_GRIPPER) == 10) {
			if
(braccio.coms_module.GetComsState(ComsModule::CommTypes::CT_BLUETOOTH) ==
Communication::S_CONNECTED)
				braccio.coms_module.SendData("Free",
ComsModule::CommTypes::CT_BLUETOOTH);
		}
		else arm.SetJointAngles(RoboticArm::JointTypes::JT_GRIPPER, 10);
	}
```

```
	else if (gripper_state == 1) {
		if (arm.GetCurrentAngles(RoboticArm::JointTypes::JT_GRIPPER) == 65) {
			if
(braccio.coms_module.GetComsState(ComsModule::CommTypes::CT_BLUETOOTH) ==
Communication::S_CONNECTED)
				braccio.coms_module.SendData("Free",
ComsModule::CommTypes::CT_BLUETOOTH);
		}
		else arm.SetJointAngles(RoboticArm::JointTypes::JT_GRIPPER, 65);
	}
}

void RobotModule::JoggingCommand(p2List<int> &list)
{
	// There are two modes of "Jogging" or manual control
	// The first one is by joint, which means that with an specified joint you
can move it
	// to the left or to the right by adding or substracting values to the
current angle.
	if (list[1] >= 0 && list[1] < 6) {
		// From the mobile, you will send 1/0/2, which means that joint will
move towards
		// the maximum or minimum angle or will remain where it is
		if (list[2] == 2) {
			arm.SetJointAngles((RoboticArm::JointTypes)list[1], 180);
		}
		else if (list[2] == 1)
arm.SetJointAngles((RoboticArm::JointTypes)list[1], 0);
		else if (list[2] == 0)
		{
			int curr_angle =
arm.GetCurrentAngles((RoboticArm::JointTypes)list[1]);
			if (curr_angle != -1)
arm.SetJointAngles((RoboticArm::JointTypes)list[1], curr_angle);
		}
	}
	// The second one is by coordenates axis XYZ, which means that with a
selected axis, you
	// will be able to move the gripper all along that axis
	else if (list[1] >= 6 && list[1] < 9) {
		//TODO: Code to move the gripper along the X, Y or Z axis

	}
}

void RobotModule::MoveCommand(p2List<int>& list)
{
	// There are three modes of automatic control
	// The first one is by joint, which means that with an specified joint
(indexed from 0 to 5), you
	// can move it to an specifed angle.
	if (list[1] >= 0 && list[1] <= 5) {
		if (list[2] == arm.GetCurrentAngles((RoboticArm::JointTypes)list[1]))
{
			if
(braccio.coms_module.GetComsState(ComsModule::CommTypes::CT_BLUETOOTH) ==
Communication::S_CONNECTED)
				braccio.coms_module.SendData("Free",
ComsModule::CommTypes::CT_BLUETOOTH);
```

```
                }
                else arm.SetJointAngles((RoboticArm::JointTypes)list[1], list[2]);
        }

        // The second one is by coordinates axis XYZ, which means that with a
selected axis, you
        // will be able to move the gripper to an specified value, all along that
axis
        else if (list[1] >= 6 && list[1] <= 8) {
                // TODO: Set of XYZ positions, and implementation of the algorithm
that allows
                // the movement of the robot with this input.

                // In case the selectes values were the same as the current ones, a
flag of no movement
                // will be sent, if the connection allows it
                if
(braccio.coms_module.GetComsState(ComsModule::CommTypes::CT_BLUETOOTH) ==
Communication::S_CONNECTED)
                        braccio.coms_module.SendData("Free",
ComsModule::CommTypes::CT_BLUETOOTH);
        }

        // The third, and last one, is by joint position, which means that you will
be able to move
        // all robot joints, by the specification of a joint configuration,
previously saved
        else if (list[1] == 9) {
                for (int i = 0; i < ELEMENTSPOS; i++) {
                        if (positions[list[2] * 6 + i] >= 0)
arm.SetJointAngles((RoboticArm::JointTypes)i, positions[list[2] * 6 + i]);
                }
        }
}

String RobotModule::BuildStringCurrentAngles()
{
        // Returns a string with the command "Response" followed by the current
        // angles of all joints, all separated by an space character
        String angles = "RESPONSE ";
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_BASE)));
        angles.concat(" ");
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_SHOULDER
)));
        angles.concat(" ");
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_ELBOW)))
;
        angles.concat(" ");
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_WRIST_VE
R)));
        angles.concat(" ");
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_WRIST_RO
T)));
        angles.concat(" ");
        angles.concat(String(arm.GetCurrentAngles(RoboticArm::JointTypes::JT_GRIPPER)
));

        return angles;
}
```

```
void RobotModule::SavePosition(p2List<int>& list)
{
	// There are four modes of managing joint positions
	// The first one consists in, given an index or name for the position,
	// the current angles will be saved
	if (list[1] == 0) {
		if (list[2] >= 0 && list[2] < NUMBERPOSITIONS) {
			for (int i = 0; i < ELEMENTSPOS; i++) {
				positions[list[2] * ELEMENTSPOS + i] =
arm.GetCurrentAngles((RoboticArm::JointTypes)i);
			}
		}
	}
	// The second one consists in, given an index or name for the position,
	// the especified angles will be saved
	else if (list[1] == 1) {
		int posTokens = 3;
		if (list[2] >= 0 && list[2] < NUMBERPOSITIONS) {
			for (int i = 0; i < ELEMENTSPOS; i++) {
				positions[list[2] * ELEMENTSPOS + i] = list[posTokens];
				posTokens++;
			}
		}
	}
	// The third one consists in the removal of all saved positions
	else if (list[1] == 2) {
		for (int i = 0; i < NUMBERPOSITIONS*ELEMENTSPOS; i++) {
			positions[i] = -1;
		}
	}
	// The fourth, and last one, consists in, given an index or name for the
position, the angles that contains will be removed
	else if (list[1] == 3) {
		if (list[2] >= 0 && list[2] < NUMBERPOSITIONS) {
			for (int i = 0; i < ELEMENTSPOS; i++) {
				positions[list[2] * ELEMENTSPOS + i] = -1;
			}
		}
	}
	// Uncomment for debugging purposes
	/*for (int i = 0; i < NUMBERPOSITIONS*ELEMENTSPOS; i++) {
		Serial.print("Position ");
		Serial.print(i);
		Serial.print(": ");
		Serial.println(positions[i]);
	}*/
}

void RobotModule::StopProgram() {

	// In order to stop the movement, the current angles are set as target
angles.
	for (int i = 0; i < 6; i++) {
		arm.SetJointAngles((RoboticArm::JointTypes)i,
arm.GetCurrentAngles((RoboticArm::JointTypes)i));
	}

}
```

A18. RoboticArm.h

```
#ifndef __ROBOTICARM__
#define __ROBOTICARM__
#include <Servo.h>
#include "Arduino.h"
#include "JointClass.h"

class RoboticArm
{
public:
	// Enumeration that contains the index of the available joint types
	enum JointTypes {
		JT_BASE, JT_SHOULDER, JT_ELBOW, JT_WRIST_VER,
		JT_WRIST_ROT, JT_GRIPPER, JT_NULL
	};
public:
	// Constructor. Needs to set a delay in ms between 10 and 30.
	RoboticArm(int step_delay = 20);

	// Initialize Braccio to default position
	void Start(bool soft_start = true);

	// Perform Braccio 1 degree movements until it reached the target angles
	void Move();

	// Sets a new delay between single movements from 10 up to 30.
	void SetDelay(int step_delay);

	// Sets the target angle of a determinate joint type ("Base", "Shoulder",
	//"Elbow", "Wrist_ver", "Wrist_rot", "Gripper")
	void SetJointAngles(JointTypes joint_type, int angle);

	// Method that returns the current angle of an specified joint
	int GetCurrentAngles(JointTypes joint_type);
private:
	// This function, used only with the Braccio Shield V4 and greater,
	// turn ON the Braccio softly and save Braccio from brokes.
	// The SOFT_START_CONTROL_PIN is used as a software PWM
	// @param soft_start_level: the minimum value is -70, default: 0
	void SoftStart(int soft_start_level);
	// Software implementation of the PWM for the SOFT_START_CONTROL_PIN, HIGH
	void SoftwarePWM(int high_time, int low_time);

private:
	// Variables initialization
	int step_delay;
	bool moving = false;
	bool send_message = false;
	Joint* base;
	Joint* shoulder;
	Joint* elbow;
	Joint* wrist_ver;
	Joint* wrist_rot;
	Joint* gripper;
	Joint* joints[JT_NULL];
};
#endif
```

A19. RoboticArm.cpp

```
#include "RoboticArm.h"
#include "BraccioControl.h"
#include "ComsModule.h"

// The default value for the soft start is 0
#define SOFT_START_LEVEL 0

// The software PWM is connected to PIN 12. You cannot use the pin 12 if you are
using
// a Braccio shield V4 or newer
#define SOFT_START_CONTROL_PIN     12

// Low and High Limit Timeout for the Software PWM
#define LOW_LIMIT_TIMEOUT 2000
#define HIGH_LIMIT_TIMEOUT 6000

RoboticArm::RoboticArm(int step_delay)
{
	// Initialization of available joints
	SetDelay(step_delay);
	base = new Joint(0, 180, 0);
	shoulder = new Joint(15, 165, 90);
	elbow = new Joint(0, 180, 90);
	wrist_ver = new Joint(0, 180, 90);
	wrist_rot = new Joint(0, 180, 90);

	//If 73 degrees are set as the maximum value, the servo will be damaged
	//gripper = new Joint(10, 73, 10);

	gripper = new Joint(10, 65, 10);
	joints[0] = base;
	joints[1] = shoulder;
	joints[2] = elbow;
	joints[3] = wrist_ver;
	joints[4] = wrist_rot;
	joints[5] = gripper;
}

void RoboticArm::Start(bool soft_start)
{
	// Calling Braccio.Start(false) the Softstart is disabled and you can use the
pin 12
	if (soft_start) {
		pinMode(SOFT_START_CONTROL_PIN, OUTPUT);
		digitalWrite(SOFT_START_CONTROL_PIN, LOW);
	}

	// Initialization pin Servo motors
	base->GetServo().attach(11);
	shoulder->GetServo().attach(10);
	elbow->GetServo().attach(9);
	wrist_ver->GetServo().attach(6);
	wrist_rot->GetServo().attach(5);
	gripper->GetServo().attach(3);

	//Move each servo motor to initial position
```

```
        for (int i = 0; i < JT_NULL; i++) {
                joints[i]->GetServo().write(joints[i]->GetCurrentAngle());

                // Uncomment for debugging purpose

                /*Serial.print("Joint ");
                Serial.print(i);
                Serial.print(": ");
                Serial.println(joints[i]->GetCurrentAngle());*/
        }

        if (soft_start) SoftStart(SOFT_START_LEVEL);
}

void RoboticArm::SetDelay(int _step_delay)
{
        // If the value is not inside the valid range, the limit values
        // will be set.
        if (_step_delay > 30) _step_delay = 30;
        if (_step_delay < 10) _step_delay = 10;
        step_delay = _step_delay;
}

void RoboticArm::SetJointAngles(JointTypes joint_type, int angle)
{
        // Uncomment for debugging purpose

        /*Serial.print("joint_type: ");
        Serial.print(joint_type);
        Serial.print("   angle: ");
        Serial.println(angle);*/

        // If the angle is not inside the range of the joint, the limit values
        // will be set.
        if (joint_type < JT_NULL && joint_type >= 0) {
                if (angle < joints[joint_type]->GetMinAngle()) angle =
joints[joint_type]->GetMinAngle();
                if (angle > joints[joint_type]->GetMaxAngle()) angle =
joints[joint_type]->GetMaxAngle();
                joints[joint_type]->SetTargetAngle(angle);

                // Uncomment for debugging purpose
                Serial.print("SET -> ");
                Serial.print(joint_type);
                Serial.print(": ");
                Serial.println(joints[joint_type]->GetCurrentAngle());
        }
}

int RoboticArm::GetCurrentAngles(JointTypes joint_type)
{
        // The return value will be -1 if the specified joint is not correct
        int ret = -1;

        if (joint_type < JT_NULL && joint_type >= 0) {
                ret = joints[joint_type]->GetCurrentAngle();
        }

        return ret;
```

```
}

void RoboticArm::Move()
{
	moving = false;
	//For each servomotor, if next degree is not the same of the previuos one,
then perform the movement
	for (int i = 0; i < JT_NULL; i++)
	{
		int target_angle = joints[i]->GetTargetAngle();
		int current_angle = joints[i]->GetCurrentAngle();
		if (target_angle != current_angle)
		{
			moving = true;
			//One step ahead
			if (target_angle > current_angle) {
				current_angle++;
			}
			//One step beyond
			if (target_angle < current_angle) {
				current_angle--;
			}
			joints[i]->SetCurrentAngle(current_angle);
			joints[i]->GetServo().write(joints[i]->GetCurrentAngle());

			// Uncomment for debugging purpose
			Serial.print("Moved--> ");
			Serial.print(i);
			Serial.print(": ");
			Serial.println(joints[i]->GetCurrentAngle());
		}
	}

	if (moving) {
		send_message = true;
	}

	// If a bluettoth connection exists, on last loop robot was moved and on
current loop nothing was moved
	if ((braccio.coms_module.GetComsState(ComsModule::CommTypes::CT_BLUETOOTH) ==
Communication::S_CONNECTED) && !moving && send_message)
	{
		// Uncomment for debugging purpose
		Serial.println("Movement finished");

		braccio.coms_module.SendData("Free",
ComsModule::CommTypes::CT_BLUETOOTH);
		send_message = false;
	}

	// Delay to let finish the little movement
	delay(step_delay);

	// Delay to reduce de velocity of the movement
	delay(50);
}

// This function, used only with the Braccio Shield V4 and greater,
// turn ON the Braccio softly and save it from brokes.
```

```
// The SOFT_START_CONTROL_PIN is used as a software PWM
// @param soft_start_level: the minimum value is -70, default value is 0
(SOFT_START_DEFAULT)
void RoboticArm::SoftStart(int soft_start_level)
{
	long int tmp = millis();
	while (millis() - tmp < LOW_LIMIT_TIMEOUT)
		SoftwarePWM(80 + soft_start_level, 450 - soft_start_level);   //the
sum should be 530usec

	while (millis() - tmp < HIGH_LIMIT_TIMEOUT)
		SoftwarePWM(75 + soft_start_level, 430 - soft_start_level); //the sum
should be 505usec

	digitalWrite(SOFT_START_CONTROL_PIN, HIGH);
}

// Software implementation of the PWM for the SOFT_START_CONTROL_PIN,HIGH
// @param high_time: the time in the logic level high
// @param low_time: the time in the logic level low
void RoboticArm::SoftwarePWM(int high_time, int low_time)
{
	digitalWrite(SOFT_START_CONTROL_PIN, HIGH);
	delayMicroseconds(high_time);
	digitalWrite(SOFT_START_CONTROL_PIN, LOW);
	delayMicroseconds(low_time);

}
```

A20. JointClass.h

```
#ifndef __JOINTCLASS__
#define __JOINTCLASS__

#include <Servo.h>
#include "Arduino.h"

class Joint {

	// Each joint represents a servomotor
	// Each joint of a robot has a minimum angle, a maximum angle and the methods
to get them
	// Each joint of a robot has a current angle, a target angle and the methods
to get and set them

public:
	Joint(int min_angle, int max_angle, int current_angle);

	// Function that returns the servo object related to the joint
	Servo GetServo();

	// Function that returns the minimum angle the joint can reach
	int GetMinAngle();

	// Function that returns the minimum angle the joint can reach
	int GetMaxAngle();

	// Function that returns the current angle of the joint
	int GetCurrentAngle();

	// Function that sets the current angle of the joint
	void SetCurrentAngle(int angle);

	// Function that returns the target angle of the joint
	int GetTargetAngle();

	// Function that sets the target angle of the joint
	void SetTargetAngle(int angle);

private:
	// Variables declaration
	Servo servo;
	int min_angle;
	int max_angle;
	int current_angle;
	int target_angle;
};

#endif
```

A21. JointClass.cpp

```
#include "JointClass.h"

Joint::Joint(int min_angle, int max_angle, int current_angle)
{
	// Assignment of the passed variables to the joint ones
	this->min_angle = min_angle;
	this->max_angle = max_angle;
	this->current_angle = current_angle;
	target_angle = current_angle;
}

Servo Joint::GetServo()
{
	return servo;
}

int Joint::GetMinAngle()
{
	return min_angle;
}

int Joint::GetMaxAngle()
{
	return max_angle;
}

int Joint::GetCurrentAngle()
{
	return current_angle;
}

void Joint::SetCurrentAngle(int angle)
{
	current_angle = angle;
}

int Joint::GetTargetAngle()
{
	return target_angle;
}

void Joint::SetTargetAngle(int angle)
{
	target_angle = angle;

}
```

A22. List.h

```
#ifndef __p2List_H__
#define __p2List_H__

#define Null 0

#define RELEASE( x )                    \
{                                       \
    if( x != Null && x != nullptr )  \
       {                                   \
              delete x;                    \
              x = nullptr;                 \
       }                                                    \
}

// Deletes an array of buffers
#define RELEASE_ARRAY( x ) \
{                               \
    if( x != nullptr )     \
       {                           \
              delete[] x;          \
              x = nullptr;         \
       }                           \
}

typedef unsigned int uint;

/**
* Contains items from double linked list
*/
template<class tdata>
struct p2List_item
{
       tdata                 data;
       p2List_item<tdata>*   next;
       p2List_item<tdata>*   prev;

       inline p2List_item(const tdata& _data)
       {
              data = _data;
              next = prev = Null;
       }

       ~p2List_item()
       {}
};

/**
* Manages a double linked list
*/
template<class tdata>
class p2List
{

public:

       p2List_item<tdata>*   start;
```

```
	p2List_item<tdata>*   end;

private:

	unsigned int  size;

public:

	/**
	* Constructor
	*/
	inline p2List()
	{
		start = end = Null;
		size = 0;
	}

	/**
	* Destructor
	*/
	~p2List()
	{
		clear();
	}

	/**
	* Get Size
	*/
	unsigned int count() const
	{
		return size;
	}

	/**
	* Add new item
	*/
	p2List_item<tdata>* add(const tdata& item)
	{
		p2List_item<tdata>*   p_data_item;
		p_data_item = new p2List_item < tdata >(item);

		if (start == Null)
		{
			start = end = p_data_item;
		}
		else
		{
			p_data_item->prev = end;
			end->next = p_data_item;
			end = p_data_item;
		}

		++size;
		return(p_data_item);
	}

	/**
	* Deletes an item from the list
	*/
```

```
bool del(p2List_item<tdata>* item)
{
	if (item == Null)
	{
		return (false);
	}

	// Now reconstruct the list
	if (item->prev != Null)
	{
		item->prev->next = item->next;

		if (item->next != Null)
		{
			item->next->prev = item->prev;
		}
		else
		{
			end = item->prev;
		}
	}
	else
	{
		if (item->next)
		{
			item->next->prev = Null;
			start = item->next;
		}
		else
		{
			start = end = Null;
		}
	}

	RELEASE(item);
	--size;
	return(true);
}

/**
* Destroy and free all mem
*/
void clear()
{
	p2List_item<tdata>*	p_data;
	p2List_item<tdata>*	p_next;
	p_data = start;

	while (p_data != Null)
	{
		p_next = p_data->next;
		RELEASE(p_data);
		p_data = p_next;
	}
	start = end = Null;
	size = 0;
}

/**
```

```
* read / write operator access directly to a position in the list
*/
tdata& operator  [](const unsigned int index)
{
	long                    pos;
	p2List_item<tdata>*   p_item;
	pos = 0;
	p_item = start;

	while (p_item != Null)
	{
		if (pos == index)
		{
			break;
		}

		++pos;
		p_item = p_item->next;
	}

	return(p_item->data);
}

/**
* const read operator access directly to a position in the list
*/
const tdata& operator  [](const unsigned int index) const
{
	long                    pos;
	p2List_item<tdata>*   p_item;
	pos = 0;
	p_item = start;

	while (p_item != Null)
	{
		if (pos == index)
		{
			break;
		}

		++pos;
		p_item = p_item->next;
	}

	return(p_item->data);
}

/**
* const read operator access directly to a position in the list
*/
const p2List<tdata>& operator +=(const p2List<tdata>& other_list)
{
	p2List_item<tdata>*   p_item = other_list.start;

	while (p_item != Null)
	{
		add(p_item->data);
		p_item = p_item->next;
	}
```

```
	return(*this);
}

/**
* const access to a node in a position in the list
*/
const p2List_item<tdata>* At(unsigned int index) const
{
	long 			pos = 0;
	p2List_item<tdata>*	p_item = start;

	while (p_item != Null)
	{
		if (pos++ == index)
			break;

		p_item = p_item->next;
	}

	return p_item;
}

/**
* access to a node in a position in the list
*/
p2List_item<tdata>* At(unsigned int index)
{
	long 			pos = 0;
	p2List_item<tdata>*	p_item = start;

	while (p_item != Null)
	{
		if (pos++ == index)
			break;

		p_item = p_item->next;
	}

	return p_item;
}

void Copy(const p2List<tdata>& list)
{
	for (int i = 0; i < list.count(); ++i)
	{
		add(list[i]);
	}
}

// Sort
int BubbleSort()
{
	int ret = 0;
	bool swapped = true;

	while (swapped)
	{
		swapped = false;
```

```
                        p2List_item<tdata>* tmp = start;

                        while (tmp != Null && tmp->next != Null)
                        {
                                ++ret;
                                if (tmp->data > tmp->next->data)
                                {
                                        SWAP(tmp->data, tmp->next->data);
                                        swapped = true;
                                }

                                tmp = tmp->next;
                        }
                }

                return ret;
        }

        /**
        * returns the first apperance of data as index (-1 if not found)
        */
        int find(const tdata& data)
        {
                p2List_item<tdata>* tmp = start;
                int index = 0;

                while (tmp != Null)
                {
                        if (tmp->data == data)
                                return(index);

                        ++index;
                        tmp = tmp->next;
                }
                return (-1);
        }

        void InsertAfter(uint position, const p2List<tdata>& list)
        {
                p2List_item<tdata>* p_my_list = At(position);
                p2List_item<tdata>* p_other_list = list.start;

                while (p_other_list != Null)
                {
                        p2List_item<tdata>* p_new_item = new
p2List_item<tdata>(p_other_list->data);

                        p_new_item->next = (p_my_list) ? p_my_list->next : Null;

                        if (p_new_item->next != Null)
                                p_new_item->next->prev = p_new_item;
                        else
                                end = p_new_item;

                        p_new_item->prev = p_my_list;

                        if (p_new_item->prev != Null)
                                p_new_item->prev->next = p_new_item;
                        else
```

```
                        start = p_new_item;

                p_my_list = p_new_item;
                p_other_list = p_other_list->next;
            }
        }

        void Pop(tdata &my_item)
        {
            my_item = At(0)->data;
            del(start);
        }
};

#endif /*__p2List_H__*/
```

Anexo B: Código implementado en Android Studio

B1. BluetoothClass.java

```
package com.upcstudios.tania.braccioapp3;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.widget.ArrayAdapter;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Set;
import java.util.UUID;
import java.util.concurrent.ConcurrentLinkedQueue;

public class BluetoothClass {

    // Public variables definition
    public boolean BTConnected = false;
    public static final ConcurrentLinkedQueue<String> messageQueue = new
ConcurrentLinkedQueue<>();
    public static final Object messageLock = new Object();
    public enum Commands {C_CONNECT, C_DISCONNECT, C_REQUEST, C_SENDTO, C_JOGGING,
C_MOVE, C_HAND, C_SAVEPOS, C_STOP, C_NULL}
    public enum Communications{COM_SERIAL, COM_BLUETOOTH,COM_NULL}
    public enum Joints{
        J_BASE, J_SHOULDER, J_ELBOW, J_WRIST_VER, J_WRIST_ROT, J_GRIPPER, J_X, J_Y,
J_Z, J_NULL
    }

    // Private variables definition
    private Context BTContext;
    private BluetoothAdapter BTAdapter = null;
    private ArrayAdapter<String> PairedDevicesArrayAdapter;
    private String BTAddress = null;
    private BluetoothSocket BTSocket = null;
    private BluetoothDevice BTDevice;
    private UUID BTMODULEUUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");
    private Thread btConnectionThread;
    private OutputStream mmOutStream = null;

    // Constructor of the class Bluetooth, which needs a context
    public BluetoothClass(Context context) {
        BTContext = context;
        BTAdapter = BluetoothAdapter.getDefaultAdapter();
        PairedDevicesArrayAdapter = new ArrayAdapter<String>(context,
R.layout.device_list_item);
    }
```

```
    // Method that returns the mobile Bluetooth adapter object
    public BluetoothAdapter GetBTAdapter() {
        return BTAdapter;
    }

    // Method that verifies if the Bluetooth Adapter is enabled. If not, it is asked
to the user
    // to be activated
    public void VerifyBT() {
        if (BTAdapter == null)
            Toast.makeText(BTContext, "This device does not support Bluetooth",
Toast.LENGTH_LONG).show();
        else {
            if (!BTAdapter.isEnabled()) {
                Intent intent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE).setFlags(Intent.FLAG_ACTIVITY_NEW_TAS
K);
                BTContext.startActivity(intent);
            }
        }
    }
    // Method that return a list of paired devices provided by the Bluetooth adapter
    public ArrayAdapter<String> GetPairedDevices() {
        if (PairedDevicesArrayAdapter.getCount() != 0)
PairedDevicesArrayAdapter.clear();
        VerifyBT();
        if (BTAdapter.isEnabled()) {
            Set<BluetoothDevice> pairedDevices = BTAdapter.getBondedDevices();
            if (pairedDevices.size() > 0) {
                for (BluetoothDevice device : pairedDevices) {
                    PairedDevicesArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
                }
            }
        }
        else Toast.makeText(BTContext, "Bluetooth adapter disabled",
Toast.LENGTH_SHORT).show();
        return PairedDevicesArrayAdapter;
    }

    // Method that sets the address of the device to be linked, and tries to set the
connection
    public void SetAddress(String address) throws IOException {
        BTAddress = address;
        BTDevice = BTAdapter.getRemoteDevice(BTAddress);

        SetSocket();
        if (BTConnected) {
            btConnectionThread = new Thread(new ConnectedRunnable(BTSocket));
            btConnectionThread.start();
        }
    }

    // Method that return the address of the linked remote device
    public String GetAddress() {
        return BTAddress;
    }
```

```
    // Method that performs the connection between the Bluetooth adapter and the
remote selected device
    private void SetSocket() throws IOException {
        try {
            BTSocket =
BTDevice.createInsecureRfcommSocketToServiceRecord(BTMODULEUUID);
            try {
                BTSocket.connect();
                ArrayList<String> array = new ArrayList<String>();
                writeMessage(Commands.C_CONNECT, Communications.COM_BLUETOOTH,
Joints.J_NULL, array);
            } catch (IOException e) {
                Toast.makeText(BTContext, "Error connecting: " + e.getMessage(),
Toast.LENGTH_SHORT).show();
                try {
                    BTSocket.close();
                } catch (IOException e2) {}
            }
            BTConnected = BTSocket.isConnected();
        } catch (IOException e) {
            Toast.makeText(BTContext, "Socket creation failed. Error: " +
e.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }

    // Method that sets the UUID
    public void SetUUID(UUID NewUUID) {
        BTMODULEUUID = NewUUID;
    }

    // Method used to close an active connection
    public void CloseConnection() {
        try {
            BTSocket.close();
            BTConnected = false;
            btConnectionThread.interrupt();
            mmOutStream = null;
        } catch (IOException e) {}
    }

    // Method that sets the input data stream and allows to read the received data
    private class ConnectedRunnable implements Runnable{
        private final InputStream mmInStream;
        //private final OutputStream mmOutStream;

        public ConnectedRunnable(BluetoothSocket btsocket){
            // Get the input and output streams, using temp objects because
            // member streams are final
            InputStream tmpIn = null;
            try {
                tmpIn = btsocket.getInputStream();
                //tmpOut = btsocket.getOutputStream();
            } catch (IOException e) {}
            mmInStream = tmpIn;
        }

        @Override
        public void run() {
            char ch; // bytes returned from read()
```

```
            String data;
            System.out.println("BTConnection: Begin Connection Thread");

            // Keep listening to the InputStream until an exception occurs
            while (true) {
                try {
                    data = "";
                    // Read from the InputStream
                    while ((ch = (char) mmInStream.read()) != '#') {
                        data = data + ch;
                    }

                    // Get lock and add new item
                    synchronized (messageLock) {
                        messageQueue.add(data);
                        messageLock.notifyAll();
                    }
                } catch (IOException e) {
                    break;
                }
            }
        }
    }

    // Method that allows to send a coded command
    public void writeMessage(Commands command, Communications communication, Joints
joint, ArrayList<String> arguments) {
        try {
            if(BTSocket != null && mmOutStream == null)
                mmOutStream = BTSocket.getOutputStream();
            String data = ProcessCommands(command, communication, joint, arguments);
            if (data != "") {
                data = data.concat("#");
                mmOutStream.write(data.getBytes());
            }
        } catch (IOException e) {
            System.out.println("BTConnection: Error writting to BT");
        }
    }
    // Method that allows to send a string
    public void writeMessage(String data) {
        try {
            if(BTSocket != null && mmOutStream == null)
                mmOutStream = BTSocket.getOutputStream();
            if (data != "") {
                data = data.concat("#");
                mmOutStream.write(data.getBytes());
            }
        } catch (IOException e) {
            System.out.println("BTConnection: Error writting to BT");
        }
    }

    // Method used to code commands sent form the activities, in order to sent the
result to the
    // Arduino.
    public String ProcessCommands (Commands command, Communications communication,
Joints joint, ArrayList<String> arguments) {
        String data = "";
```

```
int arg1 = 0;
int arg2 = 0;
switch (command){
    case C_CONNECT: data = "0 1";
        break;
    case C_DISCONNECT: data = "1 1";
        break;
    case C_REQUEST: data = "2 1";
        break;
    case C_SENDTO: //SENDTO [COMMUNICATION, TEXT]
        switch (communication)
        {
            case COM_SERIAL:
                data = "3 0 ";
                data = data.concat(arguments.get(0));
                break;
            case COM_BLUETOOTH:
                data = "3 1 ";
                data = data.concat(arguments.get(0));
                break;
            case COM_NULL:
                break;
            default: break;
        }
        break;
    case C_JOGGING:
        arg1 = Integer.parseInt(arguments.get(0));
        if(arg1 >= 0 && arg1 <= 2) {
            switch (joint) {
                case J_BASE:
                    data = data.concat("4 0 " + arguments.get(0));
                    break;
                case J_SHOULDER:
                    data = data.concat("4 1 " + arguments.get(0));
                    break;
                case J_ELBOW:
                    data = data.concat("4 2 " + arguments.get(0));
                    break;
                case J_WRIST_VER:
                    data = data.concat("4 3 " + arguments.get(0));
                    break;
                case J_WRIST_ROT:
                    data = data.concat("4 4 " + arguments.get(0));
                    break;
                case J_GRIPPER:
                    data = data.concat("4 5 " + arguments.get(0));
                    break;
                case J_X:
                    data = data.concat("4 6 " + arguments.get(0));
                    break;
                case J_Y:
                    data = data.concat("4 7 " + arguments.get(0));
                    break;
                case J_Z:
                    data = data.concat("4 8 " + arguments.get(0));
                    break;
                case J_NULL: break;
                default: break;
            }
```

```
                }
                break;
            case C_MOVE:
                switch (joint) {
                    case J_BASE:
                        data = data.concat("5 0 " + arguments.get(0));
                        break;
                    case J_SHOULDER:
                        data = data.concat("5 1 " + arguments.get(0));
                        break;
                    case J_ELBOW:
                        data = data.concat("5 2 " + arguments.get(0));
                        break;
                    case J_WRIST_VER:
                        data = data.concat("5 3 " + arguments.get(0));
                        break;
                    case J_WRIST_ROT:
                        data = data.concat("5 4 " + arguments.get(0));
                        break;
                    case J_GRIPPER:
                        data = data.concat("5 5 " + arguments.get(0));
                        break;
                    case J_NULL:
                        if (arguments.get(0).equals("X")){
                            data = data.concat("5 6 " + arguments.get(1));
                        }
                        else if (arguments.get(0).equals("Y")){
                            data = data.concat("5 7 " + arguments.get(1));
                        }
                        else if (arguments.get(0).equals("Z")){
                            data = data.concat("5 8 " + arguments.get(1));
                        }
                        else if (arguments.get(0).equals("P")){
                            data = data.concat("5 9 " + arguments.get(1));
                        }
                        break;
                    default: break;
                }
                break;
            case C_HAND:
                data = data.concat("6 " + arguments.get(0));
                break;
            case C_SAVEPOS:
                if (arguments.get(0).equals("0") || arguments.get(0).equals("3"))
data = data.concat("7 " + arguments.get(0) + " " + arguments.get(1));
                else if (arguments.get(0).equals("1")) data = data.concat("7 1 " +
arguments.get(1) + arguments.get(2));
                else if (arguments.get(0).equals("2")) data = data.concat("7 " +
arguments.get(0));
                break;
            case C_STOP:
                data = data.concat("8");
                break;
            case C_NULL: break;
            default: break;
        }
        return data;
    }
}
```

B2. GlobalCasses.java

```
package com.upcstudios.tania.braccioapp3;

import android.app.Application;

import java.util.ArrayList;
import java.util.Arrays;

public class GlobalClasses extends Application {

    // Variable definition
    public BluetoothClass MyBluetooth;
    public ArrayList<String> CommandArray = new ArrayList<>();
    public String Code = "";
    public  int maxPositions = 20;
    public int positions[] = new int[6*maxPositions];


    @Override
    public void onCreate() {
        super.onCreate();
        MyBluetooth = new BluetoothClass(getApplicationContext());

        // Initialization of the positions available
        Arrays.fill(positions,-1);

    }

}
```

B3. DeviceList.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:textColor="@android:color/black"
    android:textSize="18sp">


</TextView>
```

B4. LinkActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.upcstudios.tania.braccioapp3.LinkActivity">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:background="@color/colorAccent"
            android:gravity="center_horizontal"
            android:text="List of devices"
            android:textColor="@android:color/black"
            android:textSize="18sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ListView
            android:id="@+id/listDevices"
            android:layout_width="250dp"
            android:layout_height="250dp" />

        <Button
            android:id="@+id/buttonSearchPaired"
            android:layout_width="225dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:gravity="center"
            android:text="Search paired devices" />
    </LinearLayout>

</android.support.constraint.ConstraintLayout>
```

B5. LinkActivity.java

```
package com.upcstudios.tania.braccioapp3;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;

public class LinkActivity extends AppCompatActivity {

    // Variable definition
    GlobalClasses globalClasses;    // Variable that contains global variables
    ListView listDevices;
    Button buttonSearchPaired;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // In order to use global variables, is necessary to get a reference to the
application class
        globalClasses = (GlobalClasses) getApplication();

        setContentView(R.layout.activity_link);

        // Link of local variables to the objects in LinkActivity screen
        listDevices = findViewById(R.id.listDevices);
        buttonSearchPaired = findViewById(R.id.buttonSearchPaired);

        // When this button is pressed, the method GetPairedDevices() of the
bluetooth class is called
        // and the data of the list is refreshed. Also, it checks if the Bluetooth
adapter is enabled,
        // and if it is not, it will ask for enabling it.
        buttonSearchPaired.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

listDevices.setAdapter(globalClasses.MyBluetooth.GetPairedDevices());
            }
        });

        listDevices.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {
                String info = ((TextView) view).getText().toString();
                String address = info.substring(info.length() - 17);
                String name = info.substring(0, info.length() - 18);
```

```
                // When an item of the list is selected, the name of the item is
verified in order
                // to work only with de HC-06 module. If it's not the case, a
message ask for
                // choosing HC-06. If it's correct, the address of the bluetooth is
set into the
                // bluetooth class, which will try to establish the connection.
                if (name.equals("HC-06")){
                    try {
                        globalClasses.MyBluetooth.SetAddress(address);

                        // If the connection is established, a message of connected
will be shown
                        // and the MainActivity will be opened.
                        if (globalClasses.MyBluetooth.BTConnected) {
                            Toast.makeText(LinkActivity.this, "Connected.",
Toast.LENGTH_SHORT).show();
                            Intent intent = new Intent(LinkActivity.this,
MainActivity.class);
                            startActivity(intent);
                        }
                    } catch (IOException e) {
                    }
                }
                else Toast.makeText(LinkActivity.this, "Select the robot device.",
Toast.LENGTH_SHORT).show();
            }
        });
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Checks if the bluetooth adapter of the mobile is turned on.
        // If it's not, the LinkApplication will ask for enabling it.
        if (!globalClasses.MyBluetooth.GetBTAdapter().isEnabled())
            globalClasses.MyBluetooth.VerifyBT();

        // If it's enabled, the list of paired devices will be loaded.
        else {
            listDevices.setAdapter(globalClasses.MyBluetooth.GetPairedDevices());
        }
    }

}
```

B6. MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.upcstudios.tania.braccioapp3.MainActivity">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Tinkerkit Braccio Robot Control"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <ImageView
            android:id="@+id/braccioImage"
            android:layout_width="150dp"
            android:layout_height="180dp"
            android:layout_gravity="center_horizontal"
            android:layout_marginBottom="50dp"
            android:layout_marginTop="50dp"
            app:srcCompat="@mipmap/braccio" />

        <Button
            android:id="@+id/buttonJogging"
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="Jogging" />

        <Button
            android:id="@+id/buttonProgram"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
            android:text="Program" />

    </LinearLayout>


</android.support.constraint.ConstraintLayout>
```

B7. MainActivity.java

```
package com.upcstudios.tania.braccioapp3;

import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    // Variable definition
    private boolean close;       // Variable that manages if the window was closed,
for handler issues
    GlobalClasses globalClasses; // Variable that contains global variables
    Button buttonJogging;
    Button buttonProgram;
    Handler controlHandler;      // Variable that manages the received data from the
Bluetooth module

    private Thread readBluetoothThread; // Variable that is waiting for data to be
received and handled
    public boolean startedThread = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // In order to use global variables, is necessary to get a reference to the
application class
        globalClasses = (GlobalClasses) getApplication();

        setContentView(R.layout.activity_main);

        // Link of local variables to the objects in MainActivity screen
        buttonJogging = findViewById(R.id.buttonJogging);
        buttonProgram = findViewById(R.id.buttonProgram);

        readBluetoothThread = new Thread(new Runnable() {

            @Override
            public void run() {

                try {
                    while (true) {

                        String newMessage = null;

                        // Thread is waiting until the reception of data is
detected. Then, it release
                        // the message received, a come back to waiting state
                        synchronized (BluetoothClass.messageLock) {
                            BluetoothClass.messageLock.wait();
```

```
                            if (startedThread) newMessage =
BluetoothClass.messageQueue.poll();
                        }

                        // SAFE ZONE START: this part of the thread can be modified
                        if (startedThread)
                        {
                            // The received message is sent to the handler to be
processed.
                            controlHandler.obtainMessage(0, newMessage.length(),-1,
newMessage).sendToTarget();
                        }
                        // SAFE ZONE END

                    }
                } catch (InterruptedException e) {
                    // In case of having an error while receiving data, this message
will be shown.
                    System.out.println("Interrupted BT read thread.");
                }
            }
        });

        controlHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);

                // This part is implemented for future issues
                if (msg.what == 0) {
                    String readMessage = (String) msg.obj;
                    // If something is needed to be done with the message, write it
here

                }
            }
        };

        // By pressing this button, the JoggingActivity will be opened.
        buttonJogging.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                close = false;
                Intent intent = new Intent(MainActivity.this,
JoggingActivity.class);
                startActivity(intent);
            }
        });

        // By pressing this button, the ProgramActivity will be opened.
        buttonProgram.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                close = false;
                Intent intent = new Intent(MainActivity.this,
ProgramActivity.class);
                startActivity(intent);

            }
```

```
        });
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Checks if it is the first time the user has entered to this activity, in
order to open
        // the thread just once
        if (readBluetoothThread.getState() == Thread.State.NEW)
readBluetoothThread.start();
        // startedThread variable controls that the receiving data is not processed
if it current activity is
        // not this one
        else startedThread = true;

        // close variable helps to know if the activity has to be close or not (When
navigating to
        // jogging and program activities) in order to no to close the thread until
the user comes
        // back to LinkActivity
        close = true;
    }

    @Override
    protected void onPause() {
        super.onPause();

        // The connection will be interrupted when going to the LinkActivity
        if (close) {
            ArrayList<String> array = new ArrayList<String>();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_DISCONNECT,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
            globalClasses.MyBluetooth.CloseConnection();
            readBluetoothThread.interrupt();
        }
        // If the opened activity is jogging or program, only the process of
received data will be
        // interrupted
        else startedThread = false;
    }

}
```

B8. JoggingActivty.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.upcstudios.tania.braccioapp3.JoggingActivity">

    <TabHost
        android:id="@+id/joggingTabs"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <TabWidget
                android:id="@android:id/tabs"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:clickable="false"
                android:visibility="visible" />

            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <LinearLayout
                    android:id="@+id/tab1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center_horizontal|center"
                    android:orientation="vertical">

                    <LinearLayout
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_gravity="center_horizontal"
                        android:layout_margin="10dp"
                        android:orientation="horizontal">
```

```
        <TextView
            android:id="@+id/textView6"
            android:layout_width="180dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Joint"
            android:textAlignment="gravity"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <TextView
            android:id="@+id/textView5"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Current"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="5dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/baseLabel"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Base"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <Button
            android:id="@+id/buttonJogMinus1"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlus1"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />
```

```
        <TextView
            android:id="@+id/textCurr1"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="5dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/shoulderLabel"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Shoulder"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <Button
            android:id="@+id/buttonJogMinus2"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlus2"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />

        <TextView
            android:id="@+id/textCurr2"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

    </LinearLayout>
```

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/elbowLabel"
        android:layout_width="80dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Elbow"
        android:textColor="@android:color/black"
        android:textSize="18sp" />

    <Button
        android:id="@+id/buttonJogMinus3"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_marginRight="10dp"
        android:layout_weight="1"
        android:text="-" />

    <Button
        android:id="@+id/buttonJogPlus3"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_marginLeft="10dp"
        android:layout_weight="1"
        android:text="+" />

    <TextView
        android:id="@+id/textCurr3"
        android:layout_width="70dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textAlignment="center"
        android:textColor="@android:color/black"
        android:textSize="18sp" />
</LinearLayout>

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp"
    android:orientation="horizontal">
```

```
        <TextView
            android:id="@+id/wristVerLabel"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Wrist ver."
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <Button
            android:id="@+id/buttonJogMinus4"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlus4"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />

        <TextView
            android:id="@+id/textCurr4"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="5dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/wristRotLabel"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Wrist rot."
            android:textColor="@android:color/black"
            android:textSize="18sp" />
```

```
        <Button
            android:id="@+id/buttonJogMinus5"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlus5"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />

        <TextView
            android:id="@+id/textCurr5"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textAlignment="center"
            android:textColor="@android:color/black"
            android:textSize="18sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="10dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/gripperLabel"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Gripper"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <Button
            android:id="@+id/buttonJogMinus6"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />
```

```
            <Button
                android:id="@+id/buttonJogPlus6"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:layout_marginLeft="10dp"
                android:layout_weight="1"
                android:text="+" />

            <TextView
                android:id="@+id/textCurr6"
                android:layout_width="70dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:textAlignment="center"
                android:textColor="@android:color/black"
                android:textSize="18sp" />
        </LinearLayout>

    </LinearLayout>

    <LinearLayout
        android:id="@+id/tab2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="50dp"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="10dp"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/textView2"
                android:layout_width="180dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Axis"
                android:textColor="@android:color/black"
                android:textSize="18sp" />

            <TextView
                android:id="@+id/textView3"
                android:layout_width="70dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Current"
                android:textColor="@android:color/black"
                android:textSize="18sp" />

        </LinearLayout>
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textView4"
        android:layout_width="80dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="X"
        android:textColor="@android:color/black"
        android:textSize="18sp" />

    <Button
        android:id="@+id/buttonJogMinusX"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_marginRight="10dp"
        android:layout_weight="1"
        android:text="-" />

    <Button
        android:id="@+id/buttonJogPlusX"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_marginLeft="10dp"
        android:layout_weight="1"
        android:text="+" />

    <TextView
        android:id="@+id/textCurrX"
        android:layout_width="70dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textColor="@android:color/black"
        android:textSize="18sp" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/textView7"
        android:layout_width="80dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Y"
        android:textColor="@android:color/black"
        android:textSize="18sp" />
```

```
        <Button
            android:id="@+id/buttonJogMinusY"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlusY"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />

        <TextView
            android:id="@+id/textCurrY"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textColor="@android:color/black"
            android:textSize="18sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="5dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="5dp"
        android:layout_weight="1"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView8"
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Z"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

        <Button
            android:id="@+id/buttonJogMinusZ"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:text="-" />

        <Button
            android:id="@+id/buttonJogPlusZ"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="+" />
```

```
            <TextView
                android:id="@+id/textCurrZ"
                android:layout_width="70dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:textColor="@android:color/black"
                android:textSize="18sp" />
        </LinearLayout>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/tab3"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <EditText
            android:id="@+id/positionSelectedText"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:layout_marginTop="5dp"
            android:backgroundTint="@android:color/transparent"
            android:digits="0123456789"
            android:ems="10"
            android:hint="Insert position name"
            android:inputType="text|textPersonName|number"
            android:maxLength="2" />

        <LinearLayout
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:layout_marginTop="5dp"
            android:orientation="horizontal">

            <Button
                android:id="@+id/buttonSavePosition"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_weight="1"
                android:text="Save Pos" />

            <Button
                android:id="@+id/buttonView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="View" />

            <Button
                android:id="@+id/buttonDelete"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Delete" />
        </LinearLayout>
```

```
                    <TextView
                        android:id="@+id/resultPosText"
                        android:layout_width="300dp"
                        android:layout_height="wrap_content"
                        android:layout_marginBottom="5dp"
                        android:layout_marginTop="5dp"
                        android:hint="Position data"
                        android:lines="1"
                        android:textColor="@android:color/black"
                        android:textSize="18sp" />

                    <TextView
                        android:id="@+id/positionSavedText"
                        android:layout_width="300dp"
                        android:layout_height="wrap_content"
                        android:layout_marginBottom="5dp"
                        android:layout_marginTop="5dp"
                        android:lines="3"
                        android:text="Saved positions:"
                        android:textColor="@android:color/black"
                        android:textSize="18sp" />

                </LinearLayout>

            </FrameLayout>
        </LinearLayout>
    </TabHost>

</android.support.constraint.ConstraintLayout>
```

B9. JoggingActivity.java

```
package com.upcstudios.tania.braccioapp3;

import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TabHost;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;

public class JoggingActivity extends AppCompatActivity {

    // Variable definition
    GlobalClasses globalClasses; // Variable that contains global variables
    Button buttonJogMinus1;
    Button buttonJogMinus2;
    Button buttonJogMinus3;
    Button buttonJogMinus4;
    Button buttonJogMinus5;
    Button buttonJogMinus6;
    Button buttonJogPlus1;
    Button buttonJogPlus2;
    Button buttonJogPlus3;
    Button buttonJogPlus4;
    Button buttonJogPlus5;
    Button buttonJogPlus6;
    Button buttonSavePosition;
    Button buttonView;
    Button buttonDelete;
    Button buttonJogMinusX;
    Button buttonJogMinusY;
    Button buttonJogMinusZ;
    Button buttonJogPlusX;
    Button buttonJogPlusY;
    Button buttonJogPlusZ;
    TextView textCurr1;
    TextView textCurr2;
    TextView textCurr3;
    TextView textCurr4;
    TextView textCurr5;
    TextView textCurr6;
    TextView textCurrX;
    TextView textCurrY;
    TextView textCurrZ;
    TextView positionSavedText;
    TextView resultPosText;
    EditText positionSelectedText;
    Handler joggingHandler;         // Variable that manages the received data from
the Bluetooth module
    TabHost tabs;                   // Variable that contains the tab structure
```

```
    public enum CURRENT_BUTTON {NULL, BASE_P, BASE_M, SHOULDER_P, SHOULDER_M,
ELBOW_P, ELBOW_M,
        WRIST_VER_P, WRIST_VER_M, WRIST_ROT_P, WRIST_ROT_M, GRIPPER_P, GRIPPER_M,
X_P, X_M,
        Y_P, Y_M, Z_P, Z_M}
    CURRENT_BUTTON current_button = CURRENT_BUTTON.NULL;
    ArrayList<String> array = new ArrayList<>();

    private Thread readBluetoothThread; // Variable that is waiting for data to be
received and handled

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // In order to use global variables, is necessary to get a reference to the
application class
        globalClasses = (GlobalClasses) getApplication();

        setContentView(R.layout.activity_jogging);

        // Link of local variables to the objects in MainActivity screen
        textCurr1 = findViewById(R.id.textCurr1);
        textCurr2 = findViewById(R.id.textCurr2);
        textCurr3 = findViewById(R.id.textCurr3);
        textCurr4 = findViewById(R.id.textCurr4);
        textCurr5 = findViewById(R.id.textCurr5);
        textCurr6 = findViewById(R.id.textCurr6);
        textCurrX = findViewById(R.id.textCurrX);
        textCurrY = findViewById(R.id.textCurrY);
        textCurrZ = findViewById(R.id.textCurrZ);
        positionSavedText = findViewById(R.id.positionSavedText);
        resultPosText = findViewById(R.id.resultPosText);
        buttonJogMinus1 = findViewById(R.id.buttonJogMinus1);
        buttonJogMinus2 = findViewById(R.id.buttonJogMinus2);
        buttonJogMinus3 = findViewById(R.id.buttonJogMinus3);
        buttonJogMinus4 = findViewById(R.id.buttonJogMinus4);
        buttonJogMinus5 = findViewById(R.id.buttonJogMinus5);
        buttonJogMinus6 = findViewById(R.id.buttonJogMinus6);
        buttonJogPlus1 = findViewById(R.id.buttonJogPlus1);
        buttonJogPlus2 = findViewById(R.id.buttonJogPlus2);
        buttonJogPlus3 = findViewById(R.id.buttonJogPlus3);
        buttonJogPlus4 = findViewById(R.id.buttonJogPlus4);
        buttonJogPlus5 = findViewById(R.id.buttonJogPlus5);
        buttonJogPlus6 = findViewById(R.id.buttonJogPlus6);
        buttonJogMinusX = findViewById(R.id.buttonJogMinusX);
        buttonJogMinusY = findViewById(R.id.buttonJogMinusY);
        buttonJogMinusZ = findViewById(R.id.buttonJogMinusZ);
        buttonJogPlusX = findViewById(R.id.buttonJogPlusX);
        buttonJogPlusY = findViewById(R.id.buttonJogPlusY);
        buttonJogPlusZ = findViewById(R.id.buttonJogPlusZ);
        buttonSavePosition = findViewById(R.id.buttonSavePosition);
        buttonView = findViewById(R.id.buttonView);
        buttonDelete = findViewById(R.id.buttonDelete);
        positionSelectedText = findViewById(R.id.positionSelectedText);
        tabs = findViewById(R.id.joggingTabs);
```

```
        //-------- Tab Setup -----
        tabs.setup();

        // Add tab1
        TabHost.TabSpec spec = tabs.newTabSpec("myTab1");
        spec.setContent(R.id.tab1);
        spec.setIndicator("Joint");
        tabs.addTab(spec);

        // Add tab2
        spec = tabs.newTabSpec("myTab2");
        spec.setContent(R.id.tab2);
        spec.setIndicator("XYZ");
        tabs.addTab(spec);

        // Add tab3
        spec = tabs.newTabSpec("myTab3");
        spec.setContent(R.id.tab3);
        spec.setIndicator("Position");
        tabs.addTab(spec);

        tabs.setCurrentTab(0);

        // When the position tab is selected, the saved positions are verified and
their names/indexes
        // are shown un positionSavedText
        tabs.setOnTabChangedListener(new TabHost.OnTabChangeListener() {
            @Override
            public void onTabChanged(String s) {
                positionSavedText.setText("Saved positions:");
                if("myTab3".equals(s)){
                    resultPosText.setText("");
                    for (int position = 0; position < globalClasses.maxPositions;
position++) {
                        if(globalClasses.positions[position*6] != -1){
                            positionSavedText.append(" " +
String.valueOf(position));
                        }
                    }
                }
            }
        });
        //-------------------------

        readBluetoothThread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Begin BT read thread.");
                try {
                    while (true) {

                        String newMessage;

                        // Thread is waiting until the reception of data is
detected. Then, it release
                        // the message received, a come back to waiting state
                        synchronized (BluetoothClass.messageLock) {
                            BluetoothClass.messageLock.wait();
                            newMessage = BluetoothClass.messageQueue.poll();
```

```
                        }

                        // SAFE ZONE START: this part of the thread can be modified

                        // The received message is sent to the handler to be
processed.
                        joggingHandler.obtainMessage(0, newMessage.length(),-1,
newMessage).sendToTarget();
                        // SAFE ZONE END

                    }
                } catch (InterruptedException e) {
                    // In case of having an error while receiving data, this message
will be shown.
                    System.out.println("Interrupted BT read thread.");
                }
            }
        });

        joggingHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);

                // When a message is received, the only data processed is
                // the array of current angles sent by the Arduino, which will be
displayed
                // in the related textCurrx
                if (msg.what  == 0) {
                    String readMessage = (String) msg.obj;
                    String[] tokens = readMessage.split(" ");
                    if (tokens[0].equals("RESPONSE"))
                    {
                        textCurr1.setText(tokens[1]);
                        textCurr2.setText(tokens[2]);
                        textCurr3.setText(tokens[3]);
                        textCurr4.setText(tokens[4]);
                        textCurr5.setText(tokens[5]);
                        textCurr6.setText(tokens[6]);
                    }
                    //else Do something with the message
                }
            }
        };

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
        // angle of the base joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus1.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.BASE_M;
                        array.add("1");
```

```
globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_BASE, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.BASE_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_BASE, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
        // angle of the base joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus1.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.BASE_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_BASE, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.BASE_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_BASE, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
```

```
        // angle of the shoulder joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus2.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.SHOULDER_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_SHOULDER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.SHOULDER_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_SHOULDER, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
        // angle of the shoulder joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus2.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.SHOULDER_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_SHOULDER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.SHOULDER_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_SHOULDER, array);
                        array.clear();
```

```
globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);;
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
        // angle of the elbow joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus3.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.ELBOW_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_ELBOW, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.ELBOW_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_ELBOW, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
        // angle of the elbow joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus3.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.ELBOW_P;
                        array.add("2");
```

```
globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_ELBOW, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.ELBOW_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_ELBOW, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
        // angle of the wrist_vertical joint every cycle of the Arduino's loop,
until it reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus4.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.WRIST_VER_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_VER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.WRIST_VER_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_VER, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
```

```
        // angle of the wrist_vertical joint every cycle of the Arduino's loop,
until it reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus4.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.WRIST_VER_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_VER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.WRIST_VER_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_VER, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
        // angle of the wrist_rotate joint every cycle of the Arduino's loop, until
it reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus5.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.WRIST_ROT_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_ROT, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.WRIST_ROT_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_ROT, array);
                        array.clear();
```

```
globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
        // angle of the wrist_rotate joint every cycle of the Arduino's loop, until
it reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus5.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.WRIST_ROT_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_ROT, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.WRIST_ROT_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_WRIST_ROT, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by subtracting a
degree to the current
        // angle of the gripper joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinus6.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.GRIPPER_M;
                        array.add("1");
```

```
globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_GRIPPER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.GRIPPER_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_GRIPPER, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved by adding a degree to
the current
        // angle of the gripper joint every cycle of the Arduino's loop, until it
reaches the limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlus6.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.GRIPPER_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_GRIPPER, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.GRIPPER_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_GRIPPER, array);
                        array.clear();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the X axis
having into account
        // the negative direction, until the robot reaches its limit.
```

```
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinusX.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.X_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_X, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.X_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_X, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the X axis
having into account
        // the positive direction, until the robot reaches its limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlusX.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.X_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_X, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.X_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_X, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
```

```
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the Y axis
having into account
        // the negative direction, until the robot reaches its limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinusY.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.Y_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Y, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.Y_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Y, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the Y axis
having into account
        // the positive direction, until the robot reaches its limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlusY.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.Y_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Y, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.Y_P) {
```

```
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Y, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the Z axis
having into account
        // the negative direction, until the robot reaches its limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogMinusZ.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.Z_M;
                        array.add("1");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Z, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.Z_M) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Z, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // When pressing this button, the robot will be moved along the Z axis
having into account
        // the positive direction, until the robot reaches its limit.
        // When releasing this button, the robot will stop the movement if it is
moving.
        buttonJogPlusZ.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if(!array.isEmpty()) array.clear();
                if(motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
```

```
                    if (current_button == CURRENT_BUTTON.NULL) {
                        current_button = CURRENT_BUTTON.Z_P;
                        array.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Z, array);
                    }
                } else if(motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    if(current_button == CURRENT_BUTTON.Z_P) {
                        current_button = CURRENT_BUTTON.NULL;
                        array.add("0");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_JOGGING,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_Z, array);
                        array.clear();

//globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                    }
                }
                return true;
            }
        });

        // By pressing this button, the current joint positions will be saved in the
indicated index
        // into the postions array and into the Arduino's array
        buttonSavePosition.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if a index has been selected
                if(positionSelectedText.getText().length() == 0){
                    Toast.makeText(JoggingActivity.this, "Select a name for the
current position" , Toast.LENGTH_SHORT).show();
                }
                else {
                    // Check if the selected index is in the available range
                    int position =
(Integer.parseInt(positionSelectedText.getText().toString()));
                    if(position >= 0 && position <= globalClasses.maxPositions-1){
                        position = position*6;

                        int i;
                        // There are 6 joint angles for each position
                        // Save position in the position array
                        for (i = 0; i < 6;i++){
                            if (i == 0) globalClasses.positions[position+i] =
Integer.parseInt(textCurr1.getText().toString());
                            else if (i == 1) globalClasses.positions[position+i] =
Integer.parseInt(textCurr2.getText().toString());
                            else if (i == 2) globalClasses.positions[position+i] =
Integer.parseInt(textCurr3.getText().toString());
                            else if (i == 3) globalClasses.positions[position+i] =
Integer.parseInt(textCurr4.getText().toString());
                            else if (i == 4) globalClasses.positions[position+i] =
Integer.parseInt(textCurr5.getText().toString());
                            else if (i == 5) globalClasses.positions[position+i] =
Integer.parseInt(textCurr6.getText().toString());
                        }
```

```
                        // Save position in Arduino
                        array.clear();
                        array.add("0");
                        array.add(positionSelectedText.getText().toString());

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_SAVEPOS,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
                        Toast.makeText(JoggingActivity.this, "Position saved",
Toast.LENGTH_SHORT).show();

                        // Refresh positionSavedText TextEdit
                        resultPosText.setText("");
                        positionSavedText.setText("Saved positions:");
                        for (position = 0; position < globalClasses.maxPositions;
position++) {
                            if(globalClasses.positions[position*6] != -1){
                                positionSavedText.append(" " +
String.valueOf(position));
                            }
                        }
                    }
                    else {
                        Toast.makeText(JoggingActivity.this, "Choose a position
between 0 and " + String.valueOf(globalClasses.maxPositions-1),
Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });

        // By pressing this button, the content of the selected valid position will
be shown
        buttonView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if a index has been selected
                if(positionSelectedText.getText().length() == 0){
                    Toast.makeText(JoggingActivity.this, "Select a position" ,
Toast.LENGTH_SHORT).show();
                }
                else {
                    // Check if the selected index is in the available range
                    int position =
(Integer.parseInt(positionSelectedText.getText().toString()));
                    if(position >= 0 && position < globalClasses.maxPositions){
                        position = position*6;

                        // Check if the selected valid position is empty (contains -
1 values) or has values
                        if (globalClasses.positions[position] == -1)
resultPosText.setText("Empty position");
                        else {
                            resultPosText.setText("");
                            for (int i = 0; i < 6;i++){

resultPosText.append(String.valueOf(globalClasses.positions[position+i]));
                                if (i != 5) resultPosText.append(" ");
                            }
```

```
                        }
                    }
                    else {
                        Toast.makeText(JoggingActivity.this, "Choose a position
between 0 and " + String.valueOf(globalClasses.maxPositions-1),
Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });

        // By pressing this button, the selected valid position will be removed from
the
        // positions array and from the Arduino
        buttonDelete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if a index has been selected
                if(positionSelectedText.getText().length() == 0){
                    Toast.makeText(JoggingActivity.this, "Select a position" ,
Toast.LENGTH_SHORT).show();
                }
                else {
                    // Check if the selected index is in the available range
                    int position =
(Integer.parseInt(positionSelectedText.getText().toString()));
                    if(position >= 0 && position < globalClasses.maxPositions){
                        position = position*6;

                        // Set of -1 each value of the position selected
                        if (globalClasses.positions[position] != -1) {
                            resultPosText.setText("");
                            for (int i = 0; i < 6; i++)
globalClasses.positions[position + i] = -1;
                        }

                        // Delete position from Arduino
                        array.clear();
                        array.add("3");
                        array.add(positionSelectedText.getText().toString());

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_SAVEPOS,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);

                        // Refresh positionSavedText TextEdit
                        resultPosText.setText("");
                        positionSavedText.setText("Saved positions:");
                        for (position = 0; position < globalClasses.maxPositions;
position++) {
                            if(globalClasses.positions[position*6] != -1){
                                positionSavedText.append(" " +
String.valueOf(position));
                            }
                        }
                    }
                    else {
                        Toast.makeText(JoggingActivity.this, "Choose a position
between 0 and " + String.valueOf(globalClasses.maxPositions-1),
Toast.LENGTH_SHORT).show();
```

```
                    }
                }
            }
        });
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Initialization of the JoggingmActivity thread
        readBluetoothThread.start();

        // Request to the Arduino in order to get the current joint angles
        ArrayList<String> array = new ArrayList<String>();
        globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_REQUEST,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, array);
    }

    @Override
    protected void onPause() {
        super.onPause();

        // When exiting of this activity, the ProgramActivity Thread will be
interrupted
        readBluetoothThread.interrupt();
    }

}
```

B10. ProgramActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.upcstudios.tania.braccioapp3.ProgramActivity">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <EditText
            android:id="@+id/textCodeInput"
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:backgroundTint="@android:color/transparent"
            android:ems="10"
            android:gravity="top|left"
            android:hint="Insert some code"
            android:inputType="textMultiLine"
            android:lines="6"
            android:maxLines="20"
            android:minLines="4" />

        <Button
            android:id="@+id/buttonProcess"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:layout_marginTop="5dp"
            android:text="Process Data" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:layout_marginTop="5dp"
            android:orientation="horizontal">

            <Button
                android:id="@+id/buttonSend"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Send" />

            <Button
```

```
            android:id="@+id/buttonStop"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Stop" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/buttonSave"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Save" />

        <Button
            android:id="@+id/buttonImport"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Import" />

    </LinearLayout>

    <EditText
        android:id="@+id/textFileName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginTop="5dp"
        android:layout_weight="1"
        android:backgroundTint="@android:color/transparent"
        android:ems="10"
        android:gravity="center_vertical"
        android:hint="File name"
        android:inputType="text" />

</LinearLayout>

</android.support.constraint.ConstraintLayout>
```

B11. ProgramActivity.java

```
package com.upcstudios.tania.braccioapp3;

import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;

public class ProgramActivity extends AppCompatActivity {

    // Variable definition
    GlobalClasses globalClasses;                // Variable that contains global
variables
    EditText textCodeInput;
    EditText textFileName;
    Button buttonProcess;
    Button buttonSend;
    Button buttonStop;
    Button buttonSave;
    Button buttonImport;

    int send_command_pos = 0;                   // Index of the processed commands
to be sent
    int max_lines = 100;                        // Variable that sets the maximum
lines of code allowed to be inserted.
    Handler programHandler;                     // Variable that manages the
received data from the Bluetooth module
    private Thread readBluetoothThread;         // Variable that is waiting for data
to be received and handled

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // In order to use global variables, is necessary to get a reference to the
application class
        globalClasses = (GlobalClasses) getApplication();


        setContentView(R.layout.activity_program);

        // Link of local variables to the objects in ProgramActivity screen
        textCodeInput = findViewById(R.id.textCodeInput);
```

```
        textFileName = findViewById(R.id.textFileName);
        buttonProcess = findViewById(R.id.buttonProcess);
        buttonSend = findViewById(R.id.buttonSend);
        buttonStop = findViewById(R.id.buttonStop);
        buttonSave = findViewById(R.id.buttonSave);
        buttonImport = findViewById(R.id.buttonImport);

        readBluetoothThread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Begin BT read thread.");
                try {
                    while (true) {

                        String newMessage;

                        // Thread is waiting until the reception of data is
detected. Then, it release
                        // the message received, a come back to waiting state
                        synchronized (BluetoothClass.messageLock) {
                            BluetoothClass.messageLock.wait();
                            newMessage = BluetoothClass.messageQueue.poll();
                        }

                        // SAFE ZONE START: this part of the thread can be modified

                        // The received message is sent to the handler to be
processed.
                        programHandler.obtainMessage(0, newMessage.length(),-1,
newMessage).sendToTarget();

                        // SAFE ZONE END

                    }
                } catch (InterruptedException e) {
                    // In case of having an error while receiving data, this message
will be shown.
                    System.out.println("Interrupted BT read thread.");
                }
            }
        });

        programHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);

                // When a massage is received, free flag (indicated the robot has
ended the last command)
                // is checked in order to sent the next command
                if (msg.what  == 0) {
                    String readMessage = (String) msg.obj;
                    if (readMessage.equals("Free") && buttonStop.isEnabled()) {
                        sendCommand();
                    }
                }
            }
        };
```

```
        // This method control that every time the user clicks the textCodeInput,
the inserted lines
        // will be checked in order to not to allow more than what is specified.
        // It also disables the buttonSend in case that code text is modified.
        textCodeInput.setOnKeyListener(new View.OnKeyListener() {
            @Override
            public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {

                buttonSend.setEnabled(false);

                if(keyCode == keyEvent.KEYCODE_ENTER && keyEvent.getAction() ==
keyEvent.ACTION_DOWN)
                {
                    if(((EditText)view).getLineCount() >= max_lines) return true;
                }
                return false;
            }
        });

        // When this button is pressed, the processing of the inserted code starts.
        buttonProcess.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ArrayList<String> temp = new ArrayList<>();

                // First of all, an array, whose elements are each line of the
inserted coded, is needed
                // The character that separates each line is a line break
                String codeTokens[] =
textCodeInput.getText().toString().split("\n");

                // Clear of the commandArray, in order to saved only the new valid
commands
                globalClasses.CommandArray.clear();

                // Remove empty strings of the lines array
                for (int i=0; i<codeTokens.length;i++){
                    if(codeTokens[i].length() != 0) temp.add(codeTokens[i]);
                }
                codeTokens = temp.toArray(new String[temp.size()]);

                // There will be a validation process for each line of code. When
the format of a
                // line of code is incorrect, the processing of the code will be
stopped and the
                // buttonSend will remain inactive until all the code is verified as
correct.
                // For each error found, a message with its description will be
shown
                for (int i=0; i<codeTokens.length;i++){

                    // The first check point is if the line stats with ;, which
means that is an empty line
                    if (!codeTokens[i].equals(";")){
                        // The second check point is if the code line ends with the
; character
                        if (codeTokens[i].endsWith(";")) {
```

```
                            // If the code line ends with the ; character, a new
array with the arguments
                            // of the code line can be created
                            codeTokens[i] =
codeTokens[i].substring(0,codeTokens[i].length()-1);
                            String argTokens[] = codeTokens[i].split(" ");

                            // Clear of temp arraylist for future usages
                            temp.clear();

                            // Remove empty strings of the arguments array
                            for (int j=0; j<argTokens.length;j++){
                                if(argTokens[j].length() != 0)
temp.add(argTokens[j]);
                            }
                            argTokens = temp.toArray(new String[temp.size()]);

                            // Clear of temp arraylist for future usage
                            temp.clear();

                            // Check if the first argument is a valid one
                            if (argTokens[0].equals("MOV")){

                                // MOV command has to have more than an argument
                                if (argTokens.length > 1){

                                    // Check if the second argument is any of the
available ones
                                    if (argTokens[1].startsWith("M")){

                                        // If the automatic control movement is by
joint, there has to be
                                        // three arguments
                                        if (argTokens.length  == 3){


                                            // Check if the second argument is any
of the available ones,
                                            // having into account that it has to be
a motor Mx
                                            if (argTokens[1].equals("M1")){
                                                int myNum = 0;


                                                // Check if the third argument is
valid (a number in a
                                                // valid range according to the
specified motor)
                                                try{
                                                    myNum =
Integer.parseInt(argTokens[2]);
                                                    if (myNum >= 0 && myNum <= 180){
                                                        temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_BASE, temp));
                                                    }
                                                    else{
```

```
Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            }
                                            else if (argTokens[1].equals("M2")){
                                                int myNum = 0;

                                                // Check if the third argument is
valid (a number in a

                                                // valid range according to the
specified motor)
                                                try{
                                                    myNum =
Integer.parseInt(argTokens[2]);
                                                    if (myNum >= 15 && myNum <=
165){
                                                        temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_SHOULDER, temp));
                                                    }
                                                    else{

Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                        break;
                                                    }
                                                } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            }
                                            else if (argTokens[1].equals("M3")){
                                                int myNum = 0;

                                                // Check if the third argument is
valid (a number in a

                                                // valid range according to the
specified motor)

                                                try{
                                                    myNum =
Integer.parseInt(argTokens[2]);
                                                    if (myNum >= 0 && myNum <= 180){
                                                        temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_ELBOW, temp));
```

```
                                                }
                                                else{

Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                break;
                                            }
                                        }
                                        else if (argTokens[1].equals("M4")){
                                            int myNum = 0;

                                            // Check if the third argument is
valid (a number in a
                                            // valid range according to the
specified motor)
                                            try{
                                                myNum =
Integer.parseInt(argTokens[2]);
                                                if (myNum >= 0 && myNum <= 180){
                                                    temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_WRIST_VER, temp));
                                                }
                                                else{

Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                break;
                                            }
                                        }
                                        else if (argTokens[1].equals("M5")){
                                            int myNum = 0;

                                            // Check if the third argument is
valid (a number in a
                                            // valid range according to the
specified motor)
                                            try{
                                                myNum =
Integer.parseInt(argTokens[2]);
                                                if (myNum >= 0 && myNum <= 180){
                                                    temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
```

```
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_WRIST_ROT, temp));
                                                    }
                                                    else{

Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                        break;
                                                    }
                                                } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            }
                                            else if (argTokens[1].equals("M6")){
                                                int myNum = 0;

                                                // Check if the third argument is
valid (a number in a
                                                // valid range according to the

specified motor)
                                                try{
                                                    myNum =
Integer.parseInt(argTokens[2]);
                                                    if (myNum >= 10 && myNum <= 65){
                                                        temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_GRIPPER, temp));
                                                    }
                                                    else{

Toast.makeText(ProgramActivity.this, "Invalid angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                        break;
                                                    }
                                                } catch (NumberFormatException nfe){

Toast.makeText(ProgramActivity.this, "Missing angle at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                            }
                                            else {
                                                Toast.makeText(ProgramActivity.this,
"No such motor at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                                break;
                                            }
                                        }
                                        else if (argTokens.length > 3) {
                                            Toast.makeText(ProgramActivity.this,
"Too many arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                            break;
                                        }
                                        else {
```

```
                                        Toast.makeText(ProgramActivity.this,
"Missing arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                }
                                else if (argTokens[1].equals("X")){

                                    // If the automatic control movement is by
coordinate axis X,
                                    // there has to be three arguments
                                    if (argTokens.length == 3){
                                        int myNum = 0;

                                        // Check if the third argument is valid
(it has to be a number)
                                        try{
                                            myNum =
Integer.parseInt(argTokens[2]);
                                            //TODO: Limit the values
                                            temp.add(argTokens[1]);
                                            temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                                        } catch (NumberFormatException nfe){
                                            Toast.makeText(ProgramActivity.this,
"Missing value at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                            break;
                                        }
                                    }
                                    else if (argTokens.length > 3){
                                        Toast.makeText(ProgramActivity.this,
"Too many arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                    else {
                                        Toast.makeText(ProgramActivity.this,
"Missing arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                }
                                else if (argTokens[1].equals("Y")){

                                    // If the automatic control movement is by
coordinate axis Y,
                                    // there has to be three arguments
                                    if (argTokens.length == 3){
                                        int myNum = 0;

                                        // Check if the third argument is valid
(it has to be a number)
                                        try{
                                            myNum =
Integer.parseInt(argTokens[2]);
                                            //TODO: Limit the values
                                            temp.add(argTokens[1]);
                                            temp.add(argTokens[2]);
```

```
globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                                        } catch (NumberFormatException nfe){
                                            Toast.makeText(ProgramActivity.this,
"Missing value at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                            break;
                                        }
                                    }
                                    else if (argTokens.length > 3){
                                        Toast.makeText(ProgramActivity.this,
"Too many arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                    else {
                                        Toast.makeText(ProgramActivity.this,
"Missing arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                }
                                else if (argTokens[1].equals("Z")){

                                    // If the automatic control movement is by
coordinate axis Z,
                                    // there has to be three arguments
                                    if (argTokens.length == 3){
                                        int myNum = 0;

                                        // Check if the third argument is valid
(it has to be a number)
                                        try{
                                            myNum =
Integer.parseInt(argTokens[2]);
                                            //TODO: Limit the values
                                            temp.add(argTokens[1]);
                                            temp.add(argTokens[2]);

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                                        } catch (NumberFormatException nfe){
                                            Toast.makeText(ProgramActivity.this,
"Missing value at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                            break;
                                        }
                                    }
                                    else if (argTokens.length > 3){
                                        Toast.makeText(ProgramActivity.this,
"Too many arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                    else {
                                        Toast.makeText(ProgramActivity.this,
"Missing arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                }
                                else if (argTokens[1].startsWith("P")){
```

```
                                    // If the automatic control movement is by
saved joint position,
                                    // there has to be two arguments
                                    if(argTokens.length == 2){
                                        int myNum = 0;

                                        // Check if the second argument is any
of the available ones
                                        try {
                                            myNum =
Integer.parseInt(argTokens[1].substring(1));
                                            if (myNum >= 0 && myNum <
globalClasses.maxPositions){
                                                int position;
                                                for (position = 0; position < 6;
position++){
                                                    if
(globalClasses.positions[myNum*6+position] == -1) break;
                                                }
                                                if (position < 6) {

Toast.makeText(ProgramActivity.this, "The selected position at line " +
String.valueOf(i+1) + " does not exist", Toast.LENGTH_SHORT).show();
                                                    break;
                                                }
                                                else {

temp.add(argTokens[1].substring(0,1));

temp.add(argTokens[1].substring(1));

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_MOVE, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                                                }
                                            }
                                        } catch (NumberFormatException nfe){
                                            Toast.makeText(ProgramActivity.this,
"Insert a valid position at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
                                            break;
                                        }
                                    }
                                    else if (argTokens.length > 2) {
                                        Toast.makeText(ProgramActivity.this,
"Too many arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                    else {
                                        Toast.makeText(ProgramActivity.this,
"Missing arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                        break;
                                    }
                                }
                                else{
                                    Toast.makeText(ProgramActivity.this, "The
second argument does not match with the command MOV at line " + String.valueOf(i+1),
Toast.LENGTH_SHORT).show();
```

```
                                    break;
                                }
                            }
                            else {
                                Toast.makeText(ProgramActivity.this, "Missing
arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                break;
                            }
                        }
                        else if (argTokens[0].equals("HCL")){

                            //Hand close command is a command with a single
argument
                            if (argTokens.length < 2){
                                temp.add("1");

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_HAND, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                            }
                            else {
                                Toast.makeText(ProgramActivity.this, "Too many
arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                break;
                            }
                        }
                        else if (argTokens[0].equals("HOP")){

                            //Hand open command is a command with a single
argument
                            if (argTokens.length < 2){
                                temp.add("0");

globalClasses.CommandArray.add(globalClasses.MyBluetooth.ProcessCommands(BluetoothCl
ass.Commands.C_HAND, BluetoothClass.Communications.COM_NULL,
BluetoothClass.Joints.J_NULL, temp));
                            }
                            else {
                                Toast.makeText(ProgramActivity.this, "Too many
arguments at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                                break;
                            }
                        }
                        else{
                            Toast.makeText(ProgramActivity.this, "No such
command at line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                            break;
                        }
                    }
                    else {
                        Toast.makeText(ProgramActivity.this, "Missing ; at line
" + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                        break;
                    }
                }
                else {
                    Toast.makeText(ProgramActivity.this, "Insert a command at
line " + String.valueOf(i+1), Toast.LENGTH_SHORT).show();
                    break;
```

```
                    }
                }

                // If all code lines have not been processed, the commandArray will
be cleared,
                // and the send and stop buttons will remain inactive.
                if(globalClasses.CommandArray.size() < codeTokens.length) {
                    globalClasses.CommandArray.clear();
                    if(buttonSend.isEnabled()) {
                        buttonSend.setEnabled(false);
                        buttonStop.setEnabled(false);
                    }
                }
                // If all code lines have been processed, the buttonSend will be
activated and the
                // inserted code will be saved in order to not to lose it when
moving to Main o Link
                // activities.
                else {
                    globalClasses.Code = textCodeInput.getText().toString();
                    buttonSend.setEnabled(true);
                }
            }
        });

        // When this button is pressed, it sets the index of the commandArray to the
first one, sends
        // the first processed command, activates the buttonStop, disables the
buttonSend and increment
        // by 1 the commandArray index.
        buttonSend.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                send_command_pos = 0;

globalClasses.MyBluetooth.writeMessage(globalClasses.CommandArray.get(send_command_p
os));
                send_command_pos++;
                buttonSend.setEnabled(false);
                buttonStop.setEnabled(true);
            }
        });

        // When this button is pressed, it sends the command Stop to the
microcontroller, which will end
        // the current movement of the robot. Also, a message with the line where
the program has been
        // stopped will be shown. In addiction, the index of the commandArray will
be set to be first
        // position, the buttonStop will be disabled and the buttonSend will be
activated.
        buttonStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ArrayList<String> temp = new ArrayList<>();

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_STOP,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, temp);
```

```
                Toast.makeText(ProgramActivity.this, "Program stopped at line " +
String.valueOf(send_command_pos), Toast.LENGTH_SHORT).show();
                send_command_pos = 0;
                buttonSend.setEnabled(true);
                buttonStop.setEnabled(false);
            }
        });

        // This button performs the file saving into a local text file
        buttonSave.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                // Check if some code text has been inserted
                if(textCodeInput.getText().length() == 0)
Toast.makeText(ProgramActivity.this, "Insert some code.",
Toast.LENGTH_SHORT).show();
                else {
                    // Check if the code inserted has been processed, by knowing the
state of the buttonSend,
                    // which is active if the a some code has been validated.
                    if (!buttonSend.isEnabled())
Toast.makeText(ProgramActivity.this, "Verify your code.",
Toast.LENGTH_SHORT).show();
                    else {
                        // Check if a file name has been inserted
                        if (textFileName.getText().length() == 0)
Toast.makeText(ProgramActivity.this, "Insert a file name.",
Toast.LENGTH_SHORT).show();
                        else {
                            // Check if a SdCard is present
                            boolean sdCardPresent =
android.os.Environment.getExternalStorageState().equals(android.os.Environment.MEDIA
_MOUNTED);
                            if(sdCardPresent){

                                // If the folder Codes does not exist, it will be
created.
                                // By having the path of the folder, a file with the
indicated name
                                // will be created, if it does not exist, and filled
with the current
                                // validated code, with the current saved joint
positions, including
                                // their index.
                                try {
                                    File root = new
File(Environment.getExternalStorageDirectory(), "Codes");
                                    if(!root.exists()) root.mkdirs();
                                    File gpxfile = new File(root,
textFileName.getText().toString()+".txt");
                                    FileWriter writer = new FileWriter(gpxfile);

writer.append(textCodeInput.getText().toString());
                                    writer.append("\nEND;\n");

                                    for (int position = 0; position <
globalClasses.maxPositions; position++) {
                                        for(int i = 0; i < 6; i++){
```

```
                                            if(globalClasses.positions[position*6+i]
== -1) break;
                                            else {
                                                if (i == 0)
writer.append(String.valueOf(position) + "+");

writer.append(String.valueOf(globalClasses.positions[position * 6 + i]));
                                                if (i != 5) writer.append("+");
                                                else {
                                                    if (position > 0 && position <
globalClasses.maxPositions-1) writer.append("\n");
                                                }
                                            }
                                        }
                                    }

                                    writer.flush();
                                    writer.close();
                                    Toast.makeText(ProgramActivity.this, "Program
saved in " + root.getPath(), Toast.LENGTH_SHORT).show();
                                } catch (IOException e){
                                    // In case that this process fails, a message
with the error description will be shown.
                                    Toast.makeText(ProgramActivity.this,
e.getMessage(), Toast.LENGTH_SHORT).show();
                                }
                            }
                            else {
                                Toast.makeText(ProgramActivity.this, "SD Card not
available", Toast.LENGTH_SHORT).show();
                            }
                        }
                    }
                }
            }
        });

        // This button performs the import of the data contained in a specified file
        buttonImport.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if a file name has been inserted
                if (textFileName.getText().length() == 0)
Toast.makeText(ProgramActivity.this, "Insert a valid file",
Toast.LENGTH_SHORT).show();
                else {
                    // Check if the Codes folder exists
                    File root = new File(Environment.getExternalStorageDirectory(),
"Codes");
                    if(!root.exists()) Toast.makeText(ProgramActivity.this, "The
path does not exist", Toast.LENGTH_SHORT).show();
                    else {
                        // Check if the indicated file exists
                        File gpxfile = new File(root,
textFileName.getText().toString()+".txt");

                        if(!gpxfile.exists()) Toast.makeText(ProgramActivity.this,
"The file does not exist", Toast.LENGTH_SHORT).show();
                        else {
```

```
                        // If all the filters have been passed, the saved code
will be inserted
                        // into the textCodeInput and saved into a global
variable and the saved
                        // positions, if they exists, will be loaded into the
global positions
                        // array, whose actual data will be deleted.
                        StringBuilder text = new StringBuilder();
                        try {
                            // Save and load the file code
                            BufferedReader br = new BufferedReader(new
FileReader(gpxfile));
                            String line;

                            while ((line = br.readLine()) != null){
                                text.append(line);
                                text.append("\n");
                            }

                            br.close();

                            String[] tokens = text.toString().split("\nEND;\n");
                            textCodeInput.setText(tokens[0]);
                            globalClasses.Code = tokens[0];

                            // Check if positions there are saved positions
                            if(tokens.length > 1) {
                                tokens[1] = tokens[1].substring(0,
tokens[1].length()-1).replace("+", " ");
                                tokens[1] = tokens[1].replace("\n", " ");
                                String[] angles = tokens[1].split(" ");
                                Arrays.fill(globalClasses.positions,-1);
                                try {
                                    int pos = 0;
                                    for(int i = 0; i < angles.length/7; i++){
                                        pos = Integer.parseInt(angles[i*7]);
                                        for (int y = 0; y < 6; y++){
                                            globalClasses.positions[pos*6+y] =
Integer.parseInt(angles[i*7+y+1]);
                                        }
                                    }

                                    ArrayList<String> temp = new ArrayList<>();

                                    // Delete all positions in Arduino
                                    temp.add("2");

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_SAVEPOS,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, temp);

                                    //Save imported positions into the positions
array and into the Arduino
                                    for (int position = 0; position <
globalClasses.maxPositions; position++) {
                                        if(globalClasses.positions[position*6]
!= -1){
                                            temp.clear();
                                            temp.add("1");
```

```
                                                temp.add(String.valueOf(position));
                                                String posAngles = "";
                                                for (int i = 0; i < 6; i++)
posAngles = posAngles.concat(" " +
String.valueOf(globalClasses.positions[position*6+i]));
                                                temp.add(posAngles);

globalClasses.MyBluetooth.writeMessage(BluetoothClass.Commands.C_SAVEPOS,
BluetoothClass.Communications.COM_NULL, BluetoothClass.Joints.J_NULL, temp);
                                            }
                                        }

                                        Toast.makeText(ProgramActivity.this, "New
positions saved", Toast.LENGTH_SHORT).show();
                                    } catch (NumberFormatException e){
                                        Toast.makeText(ProgramActivity.this, "Error
while loading positions", Toast.LENGTH_SHORT).show();
                                    }
                                }

                            } catch (IOException e){
                                Toast.makeText(ProgramActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
                            }
                        }
                    }
                }
            }
        });

    }

    @Override
    protected void onResume() {
        super.onResume();

        // Initialization of the ProgramActivity thread
        readBluetoothThread.start();

        // When entering to this activity, the saved code will be loaded onto de
textCodeInput and the
        // send and stop buttons will be set disabled.
        if(globalClasses.Code != ""){
            textCodeInput.setText(globalClasses.Code);
        }
        buttonSend.setEnabled(false);
        buttonStop.setEnabled(false);
    }

    @Override
    protected void onPause() {
        super.onPause();

        // When exiting of this activity, the ProgramActivity Thread will be
interrupted
        readBluetoothThread.interrupt();
    }
```

```
    // When calling this function, the next processed command will be sent and the
index of the command
    // will be incremented by 1. In case of not having next command, the index of
the commandArray
    // will be set to be first position, the buttonStop will be disabled and the
buttonSend will be activated.
    private void sendCommand(){
        if (send_command_pos < globalClasses.CommandArray.size()){

globalClasses.MyBluetooth.writeMessage(globalClasses.CommandArray.get(send_command_p
os));
            send_command_pos++;
        }
        else {
            send_command_pos = 0;
            buttonSend.setEnabled(true);
            buttonStop.setEnabled(false);
        }
    }
```